VŠB – Technická univerzita Ostrava
Fakulta elektrotechniky a informatiky
Katedra informatiky

Implementace správy reportů pro firmu Hyundai Glovis Czech Republic s. r. o.

Implementation of Report Management for Hyundai Glovis Czech Republic s. r. o. company

2020 Andrej Mikoláš

VŠB - Technická univerzita Ostrava
Fakulta elektrotechniky a informatiky
Katedra informatiky

Zadání diplomové práce

Student: **Bc. Andrej Mikoláš**

Studijní program: N2647 Informační a komunikační technologie

Studijní obor: 2612T025 Informatika a výpočetní technika

Téma: Implementace správy reportů pro firmu Hyundai Glovis Czech Republic s. r. o.
Implementation of Report Management for Hyundai Glovis Czech Republic s. r. o. company

Jazyk vypracování: slovenština

Zásady pro vypracování:

V současnosti je systém reportů založen na přímém posílání XML/CSV dokumentů emailem z několika databází či několika speciálních aplikací. Cílem práce bude analyzovat současný stav, ve spolupráci s firmou navrhnout vylepšení a implementovat nový jednotný systém. Tento systém by měl generovat reporty buď v jednotném formátu pomocí SQL dotazu, nebo v individuálně vytvořeném formátu dle požadavků. Tyto reporty pak budou přístupné zaměstnancům firmy přes webové rozhraní na základě definovaných oprávnění.

Zadání práce lze shrnout do následujících bodů:
1. Seznámení se s problematikou procesů ve firmě Hyundai Glovis Czech Republic s. r. o.
2. Získání a návrh požadavků firmy na nový informační systém.
3. Implementace generování a správy reportů (především ve formátu XLSX) v jazyce C# nad databázemi Oracle.
4. Implementace webové vrstvy v ASP.NET pro přístup zaměstnanců na základě oprávnění (možné propojení s Active Directory).
5. Testování systému, zhodnocení výsledků a úprava implementace na základě těcho výsledků a doporučení pro další vývoj systému.

Seznam doporučené odborné literatury:

[1] Prosise, Jeff. Programming Microsoft. NET. Microsoft Press, 2002.
[2] Abramson, Ian, Michael Abbey, Michael J. Corey, and Michelle Malcher. Oracle Database 11g. McGraw-Hill Professional Publishing, 2009.

Formální náležitosti a rozsah diplomové práce stanoví pokyny pro vypracování zveřejněné na webových stránkách fakulty.

Vedoucí diplomové práce: **Ing. Lumír Kojecký**

Datum zadání: 01.09.2019

Datum odevzdání: 30.04.2020

doc. Ing. Jan Platoš, Ph.D.
vedoucí katedry

prof. Ing. Pavel Brandštetter, CSc.
děkan fakulty

Prehlasujem, že som túto diplomovú prácu vypracoval samostatne. Uviedol som všetky literárne pramene a publikácie, z ktorých som čerpal.

V Ostrave 7. máj 2020 .....

Súhlasím so zverejnením tejto diplomovej práce podľa požiadaviek čl . 26, odst. 9 Študijného a skúšobného poriadku pre štúdium v magisterských programoch VŠB-TU Ostrava.

V Ostrave 7. máj 2020 .....

HYUNDAI GLOVIS
T.G. Masaryka 1108, 738 01 Frýdek-Místek
IČ: 27305450, DIČ: CZ27805450

Na tomto mieste by som veľmi rád poďakoval pánovi Ing. Lumírovi Kojeckému za odborné vedenie a ústretovosť počas konzultácii a vypracovávania tejto práce. Taktiež by som chcel poďakovať mojej rodine a známym za podporu počas celého štúdia a práce na tomto zadaní.

Abstrakt

Cieľom tejto diplomovej práce je návrh a implementácia informačného systému pre správu reportov v spoločnosti *Hyundai Glovis Czech Republic s.r.o.*. Súčasne sa práca zaoberá prieskumom technológii .NET a architektonickými vzormi pre prezentačnú vrstvu softvéru. Práca sa ďalej zaoberá prieskumom existujúcich systémov tretích strán s podobným zameraním. Výstupom práce je funkčný informačný systém umožňujúci automatické generovanie reportov a autorizovaný prístup ku vzniknutým reportom. Tento systém bude nasadený do produkčného prostredia a sprístupnený zamestnancom spoločnosti.

Kľúčové slová: webová aplikácia, informačný systém, správa reportov, ASP.NET, Web Forms, report, Hyundai, Glovis

Abstract

The aim of this diploma thesis is the design and implementation of an information system for report management in the *Hyundai Glovis Czech Republic s.r.o.* company. This thesis also deals with the research of .NET technology and architectural patterns for the presentation layer of software. The thesis is also focused on survey of existing third party systems with a similar functionality. The output of the diploma thesis is a fully working information system able to automatically generate reports and provide authorized access to the generated reports. This system will be deployed in a production environment and be available to company employees.

Keywords: web application, information system, reports management, ASP.NET, Web Forms, report, Hyundai, Glovis

Obsah

Zoznam použitých skratiek a symbolov

AD	–	Active Directory
API	–	Application Programming Interface (rozhranie pre programovanie aplikácii)
ASP	–	Active Server Pages
BR	–	Business Requirement (biznis požiadavka)
CRUD	–	Create, Retrieve, Update, Delete (vytvoriť, získať, aktualizovať, zmazať)
CSS	–	Cascading Style Sheets (kaskádové štýly)
CSV	–	Comma-separated values (hodnoty oddelené čiarkami)
FR	–	Functional Requirements (funkčné požiadavky)
GUID	–	Globally Unique Identifier (globálne unikátny identifikátor)
HTML	–	Hypertext Markup Language (hypertextový značkovací jazyk)
HTTP	–	Hypertext Transfer Protocol (hypertextový prenosový protokol)
HTTPS	–	Hypertext Transfer Protocol Secure(zabezpečený hypertextový prenosový protokol)
IEEE	–	Institute of Electrical and Electronics Engineers (Inštitút pre elektrotechnické a elektronické inžinierstvo)
IIS	–	Internet Information Services
IoT	–	Internet of Things (internet vecí)
IS	–	Informačný systém
MVC	–	Model View Controller
MVVM	–	Model View ViewModel
PIN	–	Personal Identification Number (osobné identifikačné číslo)
SQL	–	Structured Query Language (štruktúrovaný vyhľadávací jazyk)
TS	–	Test Case (testovací prípad)
UC	–	Use Case (prípad užitia)
UI	–	User interface (užívateľské rozhranie)
URL	–	Uniform Resource Locator (jednotný vyhľadávač prostriedku)
VPN	–	Virtual Private Network (virtuálna privátna sieť)
XML	–	Extensible Markup Language (rozšíriteľný značkovací jazyk)

Zoznam obrázkov

Zoznam tabuliek

Zoznam výpisov zdrojového kódu

Úvod

V súčasnej dobe sme zaplavení veľkým množstvom informácii. Základná definícia slova *informácia* znie: „Informácie sú fakty, skúsenosti a vedomosti, ktoré ľudstvo zbiera, zaznamenáva, spracúva a odovzdáva ďalej.“ Každá informácia zároveň predstavuje údaj. Údaj, ktorý má pre konkrétneho príjemcu pridanú informatívnu hodnotu, sa pre neho stáva informáciou. [1] Pojem „dáta“ používame pre zachytenie stavu reality v určitom časovom okamihu, ktorý sa nedá zmeniť. Je možné len získať nové dáta, v novom časovo okamihu. Spracovaním dát vytvárame informáciu. [2]

V každej spoločnosti sa pracuje s dátami, a teda aj informáciami. Môže sa jednať napríklad o dáta o zamestnancoch, zákazníkoch, produktoch, peniazoch, atď. Pre zber, uchovávanie, spracovanie a poskytovanie informácii slúži informačný systém. Ten môže mať mnoho podôb, ako napríklad kartotéka, telefónny zoznam alebo webová aplikácia. Spoločnosti, ale aj niektorí jednotlivci používajú informačné systémy na dennej báze.

Informácie uchované v informačnom systéme môžu mať rôzne podoby. Vo väčšine takýchto systémov sú dáta uložené v databázových tabuľkách. V takejto forme sú dáta na jednej strane logicky a súvisle usporiadané, avšak na druhej strane, pre ich použitie by bolo vhodnejšie ich previesť do čitateľnejšej podoby. Človek dátam lepšie porozumie, pokiaľ majú tvar nejakého grafu, mapy alebo zjednodušenej usporiadanej tabuľky. Takýto formát dát môžeme nazvať report. Reporty teda vznikajú na základe uchovaných dát za účelom jednoduchšej práce s nimi. Pre vytváranie takýchto reportov existujú informačné systémy na to určené.

Spoločnosť *Hyundai Glovis Czech Republic s.r.o.*[1] bola založená v roku 2007, má približne 600 zamestnancov a je členom holdingu *HYUNDAI MOTOR GROUP*. Pre výrobcu automobilov *Hyundai Motor Manufacturing Czech s.r.o.* zabezpečuje plánovanie a prípravu procesov pred výrobou, dopravu a obchodnú logistiku. Taktiež poskytuje výrobné a prepravné logistické služby pre približne 40 zákazníkov z Českej republiky, Kórei, Maďarska, Nemecka, Poľska, Rakúska, Slovenska a Slovinska. Aj táto spoločnosť teda pracuje s nemalým objemom rôznych dát.

Významná časť prvej kapitoly je venovaná technológii .NET, aby čitateľ získal základný prehľad v tejto oblasti a zároveň popisuje základné princípy zabezpečenia implementované v tejto technológii. V neskorších častiach práce budú využívané pojmy vysvetlené v tejto kapitole. Kapitola taktiež predstavuje najznámejšie súčasné technológie používané pre vývoj webových aplikácii. Záver kapitoly je venovaný prehľadu architektonických vzorov pre prezentačnú vrstvu softvéru.

V druhej kapitole predstavím spôsob, akým spoločnosť *Hyundai Glovis* v súčasnosti spravuje interné reporty. Popíšem spôsob ich generovania a tiež ako sa reporty následne distribuujú zamestnancom spoločnosti. Zároveň uvediem aké hlavné nedostatky tento pôvodný systém má a prečo má spoločnosť potrebu nového riešenia. V kapitole tiež predstavím prehľad existujúcich

[1] `https://www.glovis.cz/`

systémov podobného zamerania od iných dodávateľov. Z tohto prieskum by sme mali vyvodiť, či niektoré existujúce riešenie by mohlo byť využité pre potreby tejto spoločnosti.

Celá nasledujúca kapitola bude patriť návrhu nového systému podľa požiadaviek zadávateľa. Kapitola bude približne štruktúrovaná podľa normovaných šablón pre návrh softvéru. V úvode kapitoly popíšem, ako si zadávateľ predstavuje, že bude výsledný systém fungovať. Následne rozpíšem podrobnejšie, ako bude požadovaná funkcionalita poskytovaná, napríklad pomocou funkčných a kvalitatívnych požiadaviek. V závere kapitoly navrhnem architektúru systému, rozdelím ho do logických celkov a zanalyzujem, ako budú jednotlivé prípady užitia implementované.

V predposlednej kapitole priblížim implementačné detaily vybraných častí systému. Tento popis bude doplnený o ukážky zdrojových kódov, aby mal technicky skúsenejší čitateľ jasnejšiu predstavu. Uvediem tiež, ktoré knižnice tretích strán boli zakomponované pre uľahčenie implementácie.

Záverečná kapitola sa bude venovať rôznym typom automatického testovania, použitého pri tomto systéme. Popíšem tiež, aké testovacie nástroje a knižnice boli použité. Zároveň bude popísané prostredie a spôsob nasadenia systému do produkčného prostredia spoločnosti.

1 Prieskum trhu

V úvode práce sa zameriam na aktuálne trendy technológii používaných pre vývoj webových aplikácii. Keďže v zadaní práce je požadované pre implementáciu použiť jazyk C# a technológiu .NET, v tejto kapitole sa tiež zameriam na túto oblasť a vysvetlím niekoľko základných aspektov.

1.1 Webové aplikácie

Webová aplikácia predstavuje softvér, ktorý beží na webovom serveri. Na rozdiel od bežných desktop aplikácii, ktoré sú spúšťané operačným systémom, sú webové aplikácie spúšťané prostredníctvom webového prehliadača. Pre vývojárov predstavujú webové aplikácie výhodu v tom, že nemusia vyvíjať jeden softvér pre viacero platforiem, ako je to v prípade iných typov aplikácii. Ďalšou výhodou je, že po aktualizácii webovej aplikácie sa všetkým užívateľom sprístupnia nové funkcionality bez potreby jej opätovnej distribúcie. Medzi ďalšie výhody patrí napríklad jednotné užívateľské rozhranie naprieč platformami, prenositeľnosť dát medzi zariadeniami alebo šetrenie výkonu klientskeho zariadenia. Medzi nevýhody by sme mohli zaradiť napríklad obmedzený prístup ku systémovým zdrojom (pamäť, procesor), pretože nebežia priamo na klientskom operačnom systéme. [3]

Webový framework

Predstavuje kolekciu knižníc predpripraveného kódu pre potreby vývoja dynamických webových aplikácii. Pomocou webového frameworku môžeme vytvárať webové služby, aplikácie alebo API, ktoré môžu byť distribuované prostredníctvom webu. [4]

1.1.1 Súčasné trendy vo vývoji webových aplikácii [4]

V tejto sekcii sa zameriam na najobľúbenejšie frameworky a knižnice pre rok 2020. Ich výber som uskutočnil na základe porovnaní z viacerých online zdrojov, ktoré prezentovali súčasné trendy pre vývoj webových aplikácii. U každého z nich uvediem, či sa jedná o front-end alebo back-end framework alebo knižnicu, aké jazyky využívajú a základné informácie o nich.

- Angular
  - front-end framework, JavaScript a TypeScript
  - vyvinutý spoločnosťou Google, podporuje MVC architektúru, vhodný pre real-time aplikácie, rýchlo rastúca komunita vývojárov
- React
  - front-end knižnica, JavaScript
  - vyvinutá spoločnosťou Facebook, podporuje aj mobilné webové aplikácie, môže byť integrovaný s inými knižnicami
- Vue.js

  - front-end framework, JavaScript
  - založený na MVVM, rýchly vývoj, jednoduchý na naučenie, veľké spoločnosti ako Google, Amazon a Facebook ho nepodporujú
- Django
  - back-end framework, Python
  - zameraný na opakované použitie kódu, podporuje zabezpečenie, rýchly vývoj, možnosť vytvárať API
- Ruby on Rails
  - back-end framework, Ruby
  - starší framework so širokou komunitou, podpora vývoja aplikácii založených na cloude
- Spring
  - back-end framework, Java
  - založený na MVC architektúre, výkonný a ľahko udržiavateľný
- Express.js
  - back-end framework, JavaScript
  - open-source, založený na Node.js, použiteľný pre vývoj API
- Laravel
  - back-end framework, PHP
  - podporuje MVC architektúru, open-source, nie je vhodný pre mobilné webové aplikácie

1.2 Čo je .NET

Je to bezplatná, multi-platformová, viacúčelová vývojárska platforma pre vytváranie veľkého množstva typov aplikácii. Jeho použitím je možné tvoriť webové, mobilné alebo desktop aplikácie, hry alebo softvér pre IoT. Pod pojmom „vývojárska platforma“ rozumieme sadu jazykov a knižníc. Medzi jeho významné vlastnosti patrí napríklad podpora viacerých programovacích jazykov, asynchrónne a súbežné vykonávanie alebo automatická správa pamäte pomocou garbage collectora. Je spravovaná spoločnosťou Microsoft a na vývoji sa podieľa množstvo dobrovoľných prispievateľov a organizácii. [5]

1.2.1 Používané jazyky

.NET podporuje použitie viacerých programovacích jazykov [5]:

- **C#:** jednoduchý, moderný, objektovo-orientovaný a typovo bezpečný
- **F#:** multiplatformový, open-source, funkcionálny s podporou objektovej orientácie a imperatívneho programovania

- **Visual Basic:** prístupný jazyk s jednoduchou syntaxou, typovo bezpečný a objektovo orientovaný

1.2.2 Implementácie .NET

V závislosti od platformy, pre ktorú chceme vyvíjať, existuje niekoľko implementácii .NET. Tie spolu tvoria celú architektúru .NET. Každá implementácia obsahuje nasledovné komponenty [6]:

- jedno alebo viac behových prostredí,
- knižnicu tried implementujúcu .NET Standard, ktoré môžu implementovať doplnkovú funkcionalitu,
- voliteľne môže tiež obsahovať niekoľko aplikačných frameworkov alebo ďalšie vývojové nástroje.

.NET Standard
Formálna špecifikácia základných API, ktoré sú implementované základnou triednou knižnicou (Base Class Library) konkrétnej .NET implementácie. Snahou je vytvoriť určitú uniformnosť v rámci .NET ekosystému a umožniť prenositeľnosť kódu medzi platformami. Vývojári môžu vytvárať knižnice prenositeľné naprieč .NET implementáciami. [7]
.NET Core
Multiplatformová implementácia navrhnutá pre vysokú záťaž servera a cloudu. Podporované platformy sú Windows, MacOS a Linux a plne implementuje .NET Standard. Je vhodná pre škálovateľné systémy, použitie s Docker[2] kontajnermi a tiež pre implementáciu microservices[3]. Poskytuje nasledovné frameworky [8]:

- **ASP.NET Core:** cloud aplikácie
- **Xamarin:** mobilné aplikácie
- **System.Device.GPIO:** IoT aplikácie
- **Windows Presentation Foundation a Windows Forms:** Windows aplikácie
- **ML.NET:** strojové učenie

.NET Framework
Pôvodná .NET implementácia existujúca od roku 2002. Obsahuje doplňujúce API, špecifické pre Windows, čo z neho robí ideálny nástroj pre vývoj Windows desktop aplikácii. Umožňuje však vyvíjať aj webové aplikácie a služby. Vývojári môžu konzistentným spôsobom vyvíjať webové a Windows aplikácie. Od verzie 4.5 implementuje .NET Standard. [6] Poskytuje napríklad tieto frameworky [9]:

- **Windows Presentation Foundation a Windows Forms:** Windows aplikácie
- **ASP.NET:** webové aplikácie

[2] `https://www.docker.com/`

[3] `https://microservices.io/`

- **Windows Service Applications:** Windows služby

Mono

Implementácia .NET zameraná na prípady s malým behovým prostredím, ktoré poháňa Xamarin aplikácie v systémoch Android, MacOS, iOS, tvOS a watchOS. Podporuje tiež beh hier vytvorených pomocou Unity engine[4]. Podporuje všetky aktuálne zverejnené verzie .NET Standard. Bežne je používaný s just-in-time kompilátorom, avšak podporuje aj ahead-of-time kompilátor, využívaný na platformách ako napríklad iOS. [6]

UWP

Táto implementácia je určená pre vytváranie moderných Windows aplikácii s podporou dotykového ovládania a tiež pre vývoj softvéru pre IoT zariadenia. Je navrhnutá na zjednotenie rozličných typov zariadení, pre ktoré chceme vyvíjať, ako napríklad počítače, tablety, mobilné telefóny alebo Xbox. Poskytuje služby ako napríklad centralizovaný obchod s aplikáciami alebo behové prostredie AppContainer. Pre využívanie prostriedkov zariadenia musí používateľ aplikácii udeliť povolenie. [6]

1.3 Framework ASP.NET

Webový framework pre vytváranie veľkých web stránok a webových aplikácii s podporným použitím HTML, CSS a JavaScript. Ďalej tiež podporuje vytváranie webových API a použitie real-time technológii. Patrí pod implementáciu .NET Framework a je alternatívou ku ASP.NET Core. [10]

ASP.NET Web Apps

Pre vytváranie webových aplikácii poskytuje ASP.NET štyri frameworky. Každý z nich je stabilný a vyspelý, a zdieľajú základnú funkcionalitu z .NET a ASP.NET.

- **Web Forms**
  - tvorba dynamický webov
  - možnosť použiť jednoduchý drag-and-drop spôsob štýlovania stránok použitím stoviek predpripravených komponentov a tiež event-driven model
- **ASP.NET MVC**
  - rýchla tvorba dynamických webov založených na oddelení jednotlivých vrstiev pomocou architektonického vzoru Model View Controller
  - podporuje vývoj založený na testoch (Test Driven Development)
- **ASP.NET Web Pages**
  - v kombinácii s Razor poskytuje rýchly, prístupný a ľahký spôsob kombinácie serverového kódu a HTML pre tvorbu dynamického webového obsahu
- **ASP.NET Single Page Applications**

[4] `https://unity.com/`

– webová aplikácia pozostávajúca z jednej stránky, ktorej obsah je aktualizovaný na základe interakcie užívateľa bez pravidelného obnovovania stránky
– väčšina operácii je vykonávaná na strane klienta

ASP.NET Web APIs
Framework pre jednoduché vytváranie HTTP služieb dostupných pre široký okruh klientov, ako napríklad prehliadače alebo mobilné zariadenia. Predstavuje ideálny framework pre tvorbu RESTful aplikácii v .NET Framework. [10]

ASP.NET SignalR
Knižnica uľahčujúca vývoj real-time funkcionality pre webové aplikácie. Umožňuje obojsmernú komunikáciu medzi serverom a klientom. Server dokáže okamžite rozposlať požadovaný obsah všetkým pripojeným klientom. Podporuje WebSockety a obsahuje API pre správu pripojenia a autorizáciu. [10]

1.4 Bezpečnosť v ASP.NET

1.4.1 Základné pojmy [11]

Autentifikácia
Proces zisťovania identity užívateľa. Autentifikácia vo väčšine webových aplikácii je založená na informácii, že užívateľ „niečo vie“, napríklad heslo alebo PIN. Informácie použité pre identifikáciu užívateľa, napríklad meno a heslo, sa označujú ako identifikačné údaje (credentials). Tieto údaje sú získané od užívateľa a overované voči určitej autorite. Úspešne identifikovaný užívateľ je označovaný ako autentifikovaný.

Autorizácia
Proces rozhodovania, či daný užívateľ má oprávnenie pristupovať ku daným prostriedkom. Rozhodnutie môže byť učinené na základe užívateľa, jeho role alebo doplnkovej informácie o užívateľovi, tzv. claims.

Užívateľský účet
Štruktúra pre uchovávanie informácii o konkrétnom užívateľovi. V základe by mal obsahovať jeho prihlasovacie meno a heslo, ale môže zahŕňať napríklad aj mailovú adresu, dátum vytvorenia účtu, dátum posledného prihlásenia, celé meno alebo kontaktné informácie.

Užívateľská rola
Druh informácie o užívateľovi, ktorý ho zaraďuje do určitej skupiny užívateľov v kontexte zabezpečenia. V systémoch, kde je autorizácia založená na rolách užívateľa, sa takto určuje úroveň prístupu ku zabezpečeným prostriedkom.

Cookies
Dáta malej veľkosti, ktoré si webové stránky ukladajú do webového prehliadača klienta pri ich používaní. Obsahujú informácie potrebné napríklad pre uchovanie jeho autentifikačných informácii.

1.4.2 Spôsoby autentifikácie

Autentifikácia prebieha prostredníctvom určitých poskytovateľov autentifikácie (authentication providers). Sú to programové moduly schopné overiť užívateľove identifikačné údaje. V prípade .NET aplikácie je možné nastaviť typ autentifikácie v konfiguračnom súbore nasledovne: `<authentication mode= "[Windows|Forms|Passport|None]"/>` [12]

Windows Authentication

Pri tomto type je potrebné nakonfigurovať autentifikáciu aj v rámci IIS, ktoré má za úlohu poskytovať autentifikovaných užívateľov. Autentifikovaného užívateľa predstavuje objekt `WindowsPrincipal`. IIS poskytuje 4 rôzne autentifikačné metódy [13]:

- **Anonymous**
  - žiadna autentifikácia, a teda ľubovoľný užívateľ má prístup do aplikácie
- **Basic**
  - užívateľ musí zadať svoje Windows užívateľské meno a heslo
  - tieto informácie sú však posielané sieťou ako čistý text, takže tento spôsob nie je veľmi bezpečný
- **Digest**
  - rovnako ako u basic, avšak informácie sú šifrované pred posielaním cez sieť
- **Windows integrated**
  - dáta nie sú posielané cez sieť ale používa sa sieťový autentifikačný protokol Kerberos alebo NTLM (New Technology LAN Manager)
  - ak klientsky počítač patrí do domény, užívateľ nemusí zadávať svoje prihlasovacie údaje do Windows
  - najlepšia voľba pre intranetové aplikácie

Forms Authentication

Všetky neautentifikované volania sú automaticky presmerované na stránku s HTML formulárom, kde užívateľ zadá svoje prihlasovacie údaje. V prípade úspešnej autentifikácie sú vytvorené cookies, obsahujúce identifikačné údaje užívateľa, ktoré môžu byť opätovne použité pri ďalších volaniach.

Passport Authentication

Centralizovaná autentifikačná služba poskytovaná spoločnosťou Microsoft. Celý proces autentifikácie užívateľa prebieha na serveroch tejto služby. Pri neautentifikovanej požiadavke na aplikáciu je užívateľ presmerovaný na prihlasovací server a následne späť do aplikácie.

1.4.3 Spôsoby autorizácie

Autorizácia na základe užívateľa

Autorizačnými pravidlami je možné udeliť alebo zamedziť prístup konkrétnym užívateľom ku

konkrétnym URL alebo funkcionalite. Na výpise 1 je ukážka deklaratívneho nastavenia prístupu užívateľa. [14]

Autorizácia na základe rolí

Tento spôsob autorizácie prichádza do úvahy keď autorizácia na základe užívateľov s rastúcim množstvom pravidiel začína byť neprehľadná a zložitá. Je možné definovať autorizačné pravidlá pre URL pre skupiny užívateľov na základe ich role. Jeden užívateľ môže mať pridelených viacero rolí. Užívateľ teda na základe rolí, ktoré má pridelené, je autorizovaný pristupovať k určeným URL. [15] Na výpise 1 je ukážka konfigurácie tohto typu autorizácie prístupu k URL. Pravidlo povoľuje vstup užívateľom s rolami *Administrators* alebo *Supervisors* a zakazuje všetkých ostatných užívateľov, okrem užívateľa John.

```
<authorization>
    <allow roles="Administrators, Supervisors" />
    <allow users="John"/>
    <deny users="*" />
</authorization>
```

Výpis 1: Ukážka konfigurácie autorizačných pravidiel na základe rolí

1.5 ASP.NET Web Forms

Časť webového frameworku ASP.NET. Samotné webové formuláre (Web Forms) predstavujú stránky vyžiadané užívateľom cez webový prehliadač. Stránky môžu byť písané kombináciou HTML, klientskych skriptov, serverových prvkov a serverového kódu. Pri požiadavke na stránku je táto kompilovaná a vykonaná na serveri pomocou frameworku, a následne je vygenerovaný HTML kód, ktorý je prehliadač schopný zobraziť. Tento kód je naviac kompatibilný s ktorýmkoľvek prehliadačom alebo mobilným zariadením.

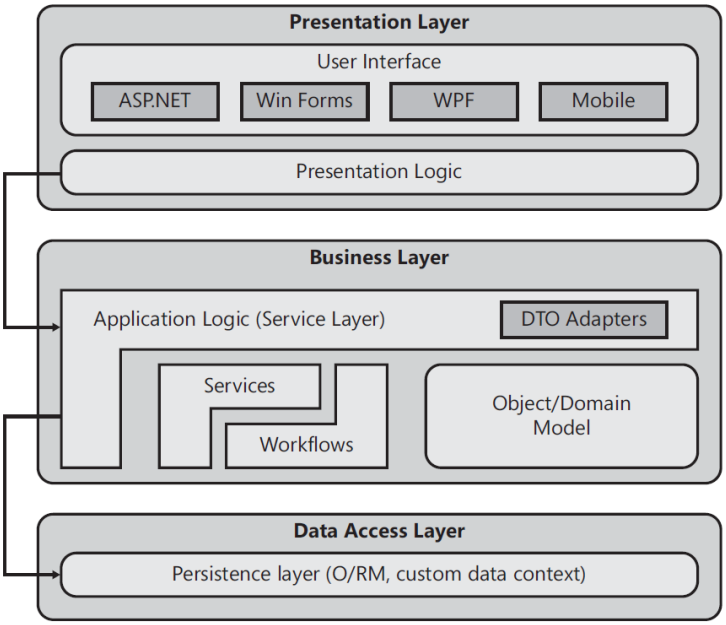
Primárne vývojové prostredie pre tento framework je Microsoft Visual Studio, ktoré umožňuje dizajnovať stránky použitím princípu drag-and-drop. Ďalej je možné jednoducho nastavovať ovládacím prvkom na stránke rôzne parametre, metódy alebo registrovať udalosti nad nimi, čím definujeme správanie a výzor stránok. Pre implementáciu prezentačnej logiky sa používa jazyk C# alebo Visual Basic. Je možné taktiež vytvárať vlastné ovládacie prvky pre stránky.

ASP.NET Web Forms umožňujú zreteľné oddelenie kódu pre užívateľské rozhranie od aplikačnej logiky, vykonávanie skriptov priamo v prehliadači klienta alebo data binding. Ponúkajú bohatú sadu serverových komponent. Podporujú tiež smerovanie, lokalizáciu stránok, testovanie, ladenie, spracovanie chýb, bezpečnosť, správu stavov a iné. [16]

1.6 Vzory pre prezentačnú vrstvu [17]

Prezentačná vrstva je jediná viditeľná vrstva, s ktorou prichádza užívateľ pri používaní softvéru do styku. Na obrázku 1 sú zobrazené tri hlavné vrstvy .NET aplikácii a ich vzájomná komuniká-

cia. Správanie a činnosť programu sú užívateľovi prezentované cez grafické alebo textové prvky v užívateľskom rozhraní. Tieto prvky poskytujú informácie, navrhujú akcie a zachytávajú užívateľovu aktivitu. Prezentačnú vrstvu tvoria dva hlavné komponenty a to užívateľské rozhranie a prezentačná logika. Akcie vykonané v užívateľskom rozhraní sa stávajú vstupom pre prezentačnú logiku.



Obr. 1: Ukážka vrstvenej architektúry .NET aplikácii [17]

Základným pravidlom pri implementácii prezentačnej vrstvy je jej jasné oddelenie od všetkého ostatného, teda od biznis vrstvy a vrstvy pre prístup ku dátam. Keďže samotná prezentačná vrstva je zložená z užívateľského rozhrania a prezentačnej logiky, aj tieto dva komponenty by mali byť jasne oddelené. Pre organizáciu prezentačnej logiky existujú tri základné triedy vzorov:

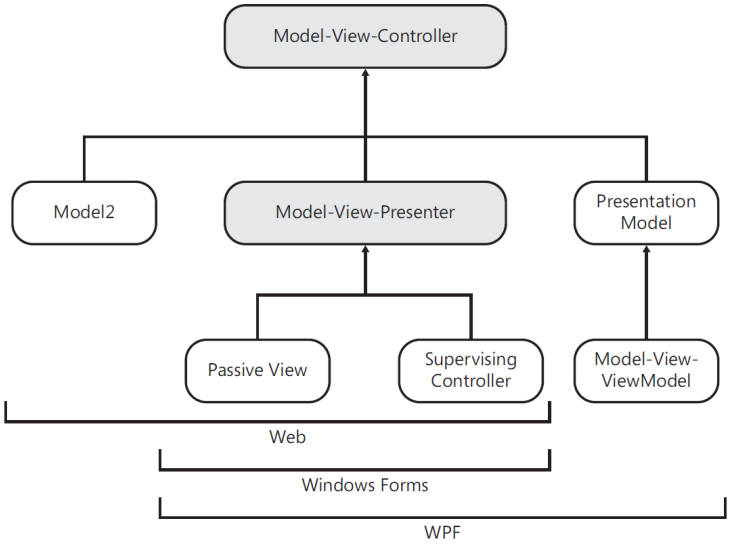
- Model View Controller (MVC),
- Model View Presenter (MVP),
- Presentation Model (PM).

Z týchto tried sú odvodené konkrétne vzory a sú zobrazené na diagrame na obrázku 2.

1.6.1 Model View Controller

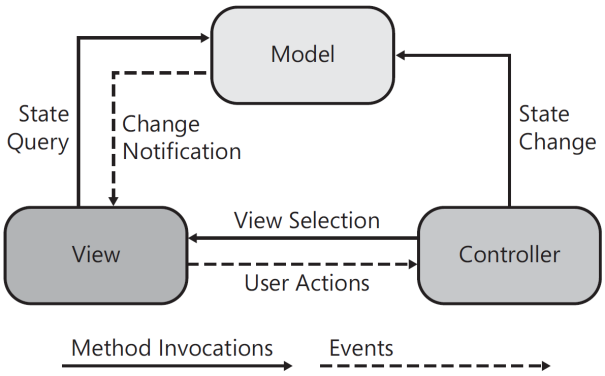
Jeho pôvodný variant bol navrhnutý ešte v roku 1979 ako náhrada vtedajších autonómnych front-end tried. Tieto triedy mali na starosti zobrazenie obsahu, udržiavanie jeho stavových informácii a obstarávanie všetkej logiky pre spracovanie užívateľských akcii. Bolo potrebné zaviesť oddelenie záujmov a znížiť previazanie medzi jednotlivými komponentami.

Dnes je MVC považovaný za vzor pre prezentačnú vrstvu, na rozdiel od jeho pôvodného účelu, kedy definoval architektúru celej aplikácie. Charakterizujú ho tri časti:



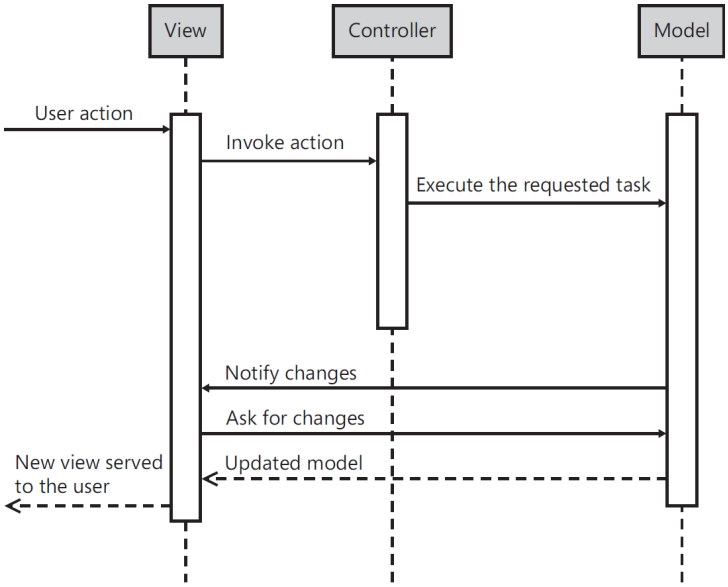
Obr. 2: Hierarchia vzorov pre prezentačnú logiku .NET aplikácií [17]

- **Model:** predstavuje dáta, s ktorými aplikácia pracuje, spravuje aplikačnú funkcionalitu a notifikuje *View* o zmenách stavu
- **View:** predstavuje obsah, ktorý sa zobrazuje užívateľovi (tlačidlá, textové polia, ...)
- **Controller:** mapuje užívateľské vstupy na akcie nad *Modelom* a aktualizuje *View*, prípadne vyberá ďalší



Obr. 3: Diagram komponent vzoru MVC [17]

Užívateľ pracuje s *View*, ktorý zachytáva jeho vstupy a predáva ich do *Controller*. Ten na základe interakcie s *Model* rozhoduje, čo robiť ďalej. *Model* nemá informácie o detaile *View*, avšak medzi týmito dvoma komponentami je vzťah „observer“. *View* si teda drží referenciu na *Model* a registruje sa u neho pre získavanie upozornení o zmenách. Pri takejto udalosti si *View* získa nové dáta z *Model* a aktualizuje užívateľské rozhranie. Táto postupnosť akcii je zachytená na sekvenčnom diagrame na obrázku 4 a komunikácia medzi jednotlivými komponentami je na obrázku 3. V niektorých implementáciách MVC je to práve *Controller*, ktorý upozorňuje *View* o zmenách.



Obr. 4: Sekvenčný diagram pôvodného vzoru MVC [17]

Model2

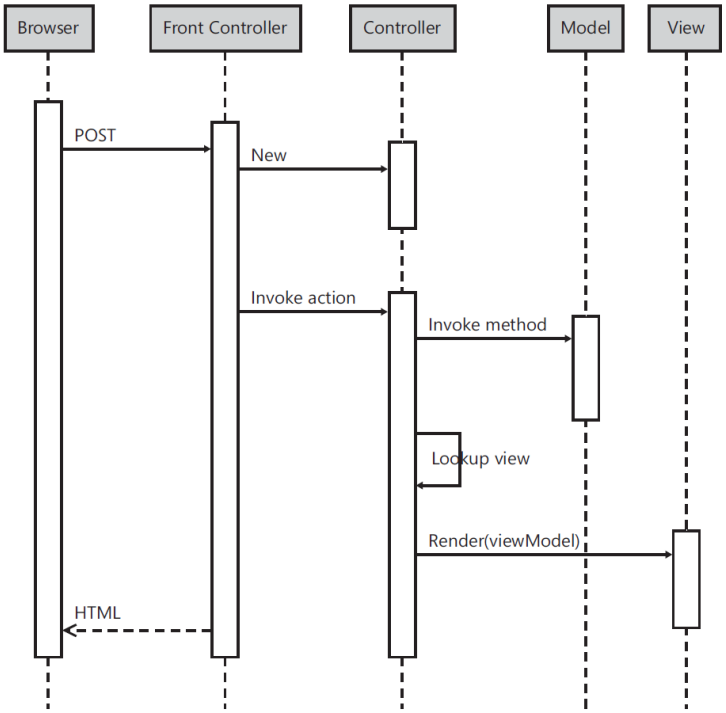
Keďže definícia pôvodnej verzie vzoru MVC bola veľmi voľná, jeho prispôsobením pre webové aplikácie vznikol variant s názvom Model2. To je jeho historický názov a dnes je základným stavebným kameňom ASP.NET MVC Frameworku, a v súčasnosti je veľmi populárny v oblasti webového vývoja. Tento vzor vystavuje *View* prostredníctvom webového prehliadača. Užívateľove akcie sú transformované na HTTP POST volania. Tie sú posielané do špeciálnej komponenty (front controller) na webovom serveri, ktorá koordinuje všetky volania do webovej aplikácie a rozhoduje, inštancia ktorého *Controllera* bude vytvorená pre spracovanie volania. Ten zavolá metódu, ktorá ovplyvní *Model* a následne upovedomí *View* aby vykreslil obsah. Sekvenčný diagram tohto vzoru je na obrázku 5.

Oproti pôvodnému MVC vzoru, nie je v tejto variante priamy kontakt medzi *View* a *Model*. Ďalším rozdielom je, že *Controller* renderuje obsah a explicitne zasiela obsah pre zobrazenie do *View*.

1.6.2 Model View Presenter

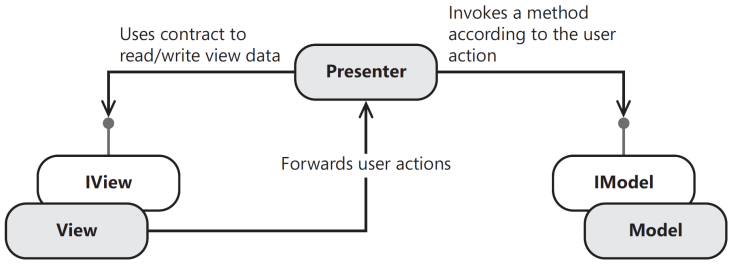
Klasický MVC vzor má dve úskalia. Za prvé, *Model* musí oznamovať *View* zmeny stavu a za druhé, *View* musí vedieť veľa o *Model*. *View* číta z *Model* ktorékoľvek dáta, ktoré potrebuje a zobrazuje ich cez vizuálne prvky. To znamená, že *View* musí obsahovať vlastnú logiku, aby sa vedel rozhodnúť, ktoré dáta z celého *Modelu* potrebuje. Táto nadbytočná réžia ho zbytočne zaťažuje.

Vhodnejšie by bolo, aby celé toto rozhodovanie mal na starosti *Controller*, aby sa *View* čo najviac odľahčil. *Controller* by teda obsahoval iba prezentačnú logiku, čo by z neho robilo ľahšie testovateľnú triedu. Pre *View* zadefinujeme určitý kontrakt (rozhranie), čo umožní ho v prípade



Obr. 5: Sekvenčný diagram vzoru Model2 [17]

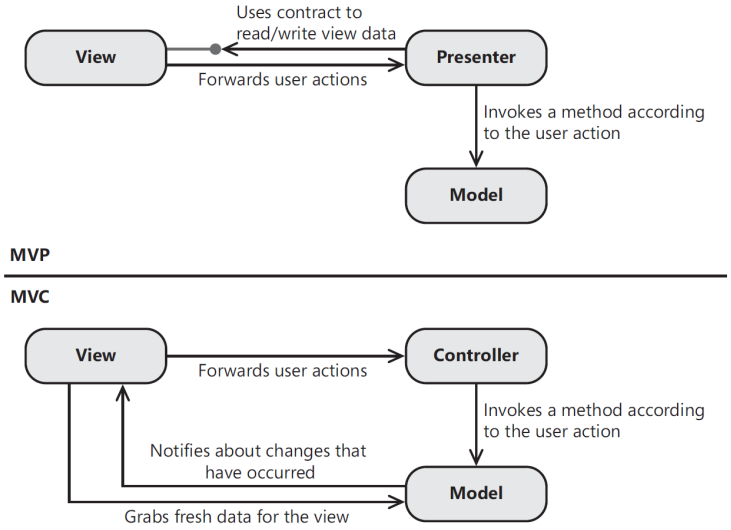
potreby vymeniť za iný, implementujúci rovnaké rozhranie. Taktiež je možné použiť existujúci *Controller* v inom systéme. Rovnaký kontrakt zadefinujeme aj pre *Model*.



Obr. 6: Schéma vzoru MVP s vystavenými rozhraniami [17]

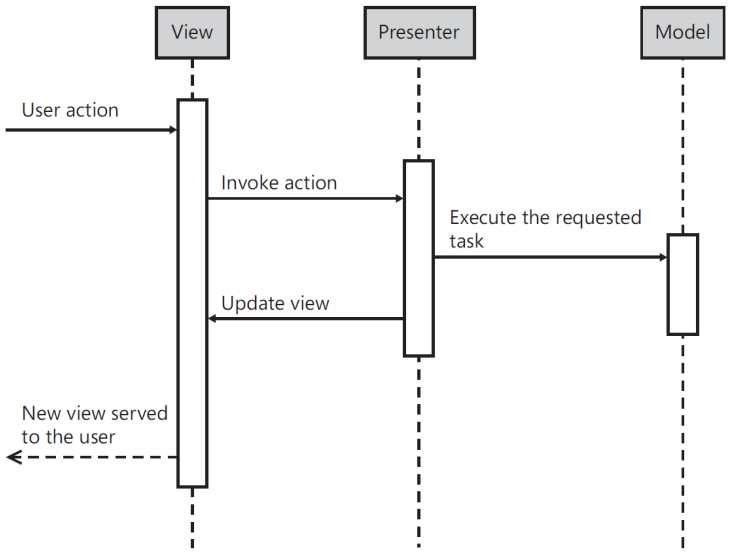
Vzor Model View Presenter je odvodený od MVC, kde bol *Controller* premenovaný na *Presenter*, pretože tento názov lepšie vystihuje jeho funkcionalitu. Na obrázku 6 je zobrazený diagram vzoru MVP spolu s rozhraniami pre *View* a *Model*. Na obrázku 7 je možné vidieť rozdiely v komunikácii komponent medzi vzorom MVP a MVC. Rozdiely v týchto vzoroch sú založené na troch faktoch:

- *View* nepozná *Model*,
- *Presenter* nepotrebuje poznať, akou technológiou je tvorené užívateľské rozhranie *View*,
- *View* môže byť nahradený zastupujúcim komponentom pre účely testovania.



Obr. 7: Porovnanie diagramov komponent vzorov MVP a MVC [17]

V prípade vzoru MVP má *View* za úlohu preposielať užívateľove vstupy do *Presenter* a zároveň od neho prijímať čerstvé dáta pre zobrazenie. *Presenter* má za úlohu obsluhovať tieto požiadavky tým, že komunikuje s *Model*. Ukážka interakcie jednotlivých komponent vzoru pri spracovávaní požiadavku je na obrázku 8.



Obr. 8: Sekvenčný diagram vzoru MVP [17]

Passive View

Komponent *View* by mal byť čo najjednoduchší a tak pasívny, ako je to možné. Tým docielime napríklad ľahšie testovateľný systém a tiež „ľahkú“ prezentačnú vrstvu. Avšak komplexný kód, ktorý odstránime z *View* sa musí presunúť inam a v tomto prípade práve do komponentu *Presenter*. Vo variante vzoru MVP *Passive View* teda máme jednoduchý *View*, ktorý sa ne-

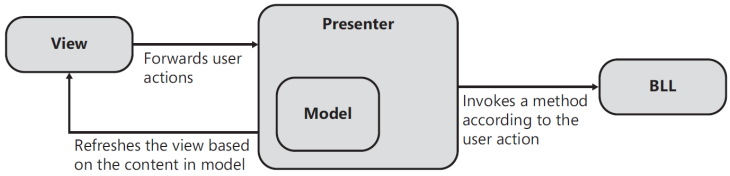
stará o formátovanie alebo data binding, avšak mierne zložitejší *Presenter*. Ten pracuje priamo s prvkami užívateľského rozhrania a vkladá dáta priamo do nich.

Supervising Controller

Ak presunieme logiku pre formátovanie a data bining do *View*, zmiernime tak výrazný rozdiel komplexnosti medzi *View* a *Presenter*. *View* sa teda musí postarať o synchronizáciu a adaptáciu vstupných dát do prvkov užívateľského rozhrania. Tento variant vzoru MVP sa nazýva *Supervising Controller*.

1.6.3 Presentation Model

Tento vzor by sa dal považovať za variantu MVP, ktorý je však vhodnejší pre zložitejšie užívateľské rozhranie. V prípade MVP, *View* vystavuje rozhranie, skrz ktoré s ním *Presenter* komunikuje. U *Presentation Model* (ďalej len PM), *View* nevystavuje žiadne rozhranie, ale dátový model je zahrnutý v *Model*, ktorý predstavuje stav *View*. Prvky vo *View* sú priamo naviazané na vlastnosti *Modelu*. Na obrázku 9 je diagram zobrazujúci komponenty vzoru PM.



Obr. 9: Diagram komponent vzoru PM [17]

Komponent *Model* je trieda s vlastnosťami pre každý nastaviteľný prvok užívateľského rozhrania vo *View*. Po inicializácii *Modelu* je *View* pripravený na vykreslenie. *Model* je aktualizovaný *Presenterom* po interakcii s aplikačnou logikou systému, a tieto dve komponenty sú zvyčajne v jednej triede typu *PresentationModel*. *View* preposiela užívateľské vstupy do *PresentationModel* a ten následne aktualizuje *Model*. Po aktualizácii *Modelu* je *View* inštruovaný aby vykreslil obsah. Je potrebné udržiavať správnu obojsmernú synchronizáciu medzi *Model* a *View*. K tomu veľmi pomáha data binding, ktorý poskytuje aj .NET framework a preto je tento vzor populárny vo Windows Presentation Foundation.

Model View ViewModel

Tento vzor je identický so spomínaným vzorom *Presentation Model*. Bol však predstavený ako štandardizovaný spôsob využitia jadrových funkcii Windows Presentation Foundation pre zjednodušenie implementácie užívateľského rozhrania.

2 Súčasný stav

V tejto kapitole sa zameriam na popis súčasného spôsobu vytvárania a distribúcie reportov v spoločnosti. Taktiež uvediem, aké sú nedostatky aktuálneho riešenia. Tento prieskum bude základom návrhu nového systému v nasledujúcej kapitole. Taktiež popíšem príklady existujúcich riešení reportovacích systémov.

2.1 Generovanie reportov

Súčasné generovanie reportov má na starosti softvérová utilita v Linuxovom systéme s názvom *Cron*. Jedná sa o plánovač úloh so širokým využitím, ktorý v stanovenom čase automaticky vykoná určitý príkaz, prípadne celý skript príkazov [18]. Všetky úlohy, ktoré má *Cron* vykonať sú udržiavané v štruktúre nazývanej „crontab“. Pre účely reportovacieho systému sú pomocou tejto utility nastavené skripty, ktoré získajú dáta z databázy, následne z nich vytvoria napríklad .csv súbor a ten na záver rozpošlú cieľovým užívateľom.

2.1.1 Štatistiky

Spoločnosť má približne 600 zamestnancov a mnoho dodávateľov, subdodávateľov a zákazníkov. V tejto skupine je približne 100 ľudí, ktorí dostávajú reporty mailom. Dáta pre reporty sú generované z troch databáz a tiež z troch ďalších aplikácii. Priemerný denný počet vygenerovaných reportov je 250 a pochádzajú z približne 95 rôznych definícii. Každý report má veľkosť pár kilobajtov. Iba minimálne množstvo z nich má veľkosť 1-3 MB a jeden má veľkosť 13MB. Tri reporty, ktoré mi boli poskytnuté ako reálne ukážky, mali veľkosti 71kB, 35kB a 16kB. Pre nasledujúce zjednodušené výpočty si vezmeme priemer týchto veľkostí. Predpokladáme teda, že veľkosť jedného reportu je $\doteq$ 40,7kB. Denne sa teda vygenerujú súbory, ktoré majú spolu veľkosť približne $250 \times 40{,}7 = 10\ 175\text{kB} \doteq 9{,}94\text{MB}$. Keďže nie každý report, ktorý sa vygeneruje, je odosielaný všetkým 100 príjemcom, predpokladajme, že sa vždy odošle priemerne 50 príjemcom. Z tohto môžeme vypočítať, že denne sa na mailovom serveri zaberie diskové miesto o veľkosti $9{,}94\text{MB} \times 50 = 497\text{MB}$. Ak počítame, že sa reporty generujú len počas pracovných dní, tak je to 2 485MB týždenne a 9 940MB $\doteq$ 9,71GB mesačne.

2.1.2 Ukážka reportu

Na obrázku 10 je časť súboru s názvom „20190225_VPC_Daily_Closing_Report.xlsx“, ako ukážka vygenerovaného reportu zo súčasného riešenia. Keďže sa jedná o firemné údaje, niektoré dáta v ukážke museli byť rozmazané. Súbor je zo dňa 25.2.2019 a jedná sa o denný uzatvárací report. Vznikol na základe dát získaných pomocou niekoľkých komplexných SQL procedúr. Ako môžeme vidieť, výsledný súbor obsahuje rôzne typy formátovania a zlučovania buniek. Takto komplexný typ reportu je výnimočný. V drvivej väčšine prípadov majú reporty podobu jednoduchej tabuľky, kde každý stĺpec je označený vlastnou hlavičkou.

HMMC Daily Closing Report

1. HMMC Production Status (TRIM IN) 24 / February. Prod. Result

Monthly Target	Production Status (Unit)			Target each model
	Daily Result	Monthly Total	Progress Rate	
[redacted]	0	[redacted]	[redacted]%	ix20 [redacted] GDe PDe [redacted] TLe [redacted]
[redacted]	0	[redacted]	[redacted]%	

2. HMMC Main Production Status

2.1 Operation Production Result (Previous day)

Classify	1st	2nd	3rd	Total	Remarks
Trim in Target	[redacted]	[redacted]	[redacted]	[redacted]	
Trim in Result				-	
Operation rate(%)	0,0%	0,0%	0,0%	0,0%	

S/OFF Target	[redacted]	[redacted]	[redacted]	[redacted]	
S/OFF Result				-	
Achiev. rate(%)	0,0%	0,0%	0,0%	0,0%	

Obr. 10: Ukážka časti vygenerovaného súboru reportu

2.2 Prístup ku reportom

Cieľovým užívateľom sa vzniknuté reporty sprístupňujú výhradne ako príloha v maile. To, ktorým užívateľom sa report odošle, je určené priamo v skripte, ktorý spúšťa plánovač úloh pri generovaní reportu. Je tiež možné uviesť ako príjemcu celú skupinu užívateľov v doméne. Mail tiež obsahuje ďalšie sprievodné informácie pre adresáta. Aby si užívateľ mohol prezrieť report, musí si ho vopred stiahnuť do počítača.

2.3 Nedostatky súčasného riešenia

Súčasné riešenie správy reportov má niekoľko hlavných nedostatkov. Prvým z nich je rýchle vyčerpávanie dostupných priestorových prostriedkov na mailovom serveri vo firemnej infraštruktúre. Ako bolo uvedené v sekcii 2.1.1, denne sa vygeneruje množstvo reportov, ktoré sa následne rozposielajú zamestnancom ako prílohy v mailových správach, a zaberajú tak miesto na mailovom serveri. Zamestnanci tak musia často archivovať staršie správy s týmito prílohami aby uvoľnili miesto, prípadne im administrátor zväčšuje pridelenú kvótu.

Ďalším nedostatkom je to, že noví zamestnanci nemajú prístup ku reportom, ktoré boli vygenerované pred tým, než nastúpili do firmy, pretože reporty sa rozposielajú ihneď po ich vygenerovaní a neukladajú sa do žiadneho centralizovaného úložiska. Poslednou nevýhodou je ťažké dohľadávanie konkrétneho reportu vo veľkom množstve obdržaných mailov. Naviac, užíva-

telia mávajú k mailovému klientovi pripojených niekoľko archívov, z ktorých každý má veľkosť cez 40GB, čo spôsobuje výrazné spomalenie vyhľadávania v mailoch.

2.4 Prehľad existujúcich riešení

V tejto sekcii popíšem existujúce riešenia od rôznych spoločností, určené pre generovanie reportov. Vzhľadom ku skutočnosti, že popisované nástroje poskytujú skutočne širokú paletu možností, v prehľade budú spomenuté len základné informácie relevantné ku zameraniu tejto práce. V závere sekcie zhrniem významné rozdiely medzi jednotlivými nástrojmi a ich výhody a nevýhody.

JReport[5]

Prepracovaný reportovací nástroj postavený na jazyku Java. Je možné ho jednoducho integrovať do front-end alebo back-end časti webových aplikácii rôzneho typu, nie len založených na jazyku Java. Umožňuje vytvárať napríklad tabuľkové, grafové alebo mapové dátové reporty. Cieľový užívateľ si môže nadefinovať aj vlastné reporty. Rôzne typy reportov sa zobrazujú na hlavnom interaktívnom paneli (dashboarde), ktorý si môže užívateľ sám prispôsobiť.

Ako zdroj dát je možné použiť viaceré typy databáz, textové alebo tabuľkové súbory, webové služby, API a iné. Reporty sú prispôsobené na prehliadanie ja na mobilných zariadeniach a podporuje mnoho výstupných formátov. Umožňuje nastaviť automatické generovanie reportov v stanovených intervaloch, verziovať ich a distribuovať mailom alebo cez zdieľané úložisko. Nástroj tiež podporuje viacero typov zabezpečení a kontrolu prístupu ku reportom.

Na webovej stránke je možné si vytvoriť demo účet za účelom vyskúšania si nástroja. Poskytovateľ systému vytvorí cenovú ponuku na základe požiadaviek zákazníka.

Fine Report[6]

Reportovací nástroj vyvinutý v jazyku Java, ktorý je možné používať ako samostatný nezávislý systém alebo ho integrovať do existujúcich webových systémov. Podporuje rôzne typy dátových zdrojov ako napríklad relačné databázy, textové súbory, NoSQL databázy a iné. V dizajnovacom nástroji podobnom bežnému tabuľkovému procesoru je možné definovať štruktúru a štýl reportov prepojením rôznych dátových zdrojov pomocou drag-and-drop prístupu. Prehliadanie reportov je prispôsobené ako pre mobilné zariadenia, tak aj pre veľké obrazovky.

Reporty sú vo forme tabuliek, širokého množstva typov grafov alebo máp založených na geografických dátach. Podporuje zabezpečený prístup ku reportom na základe oprávnení a tiež je možné nastaviť plánovač, ktorý reporty generuje v požadovaných intervaloch a následne rozposiela mailom alebo ukladá na zdieľané úložisko. Spoločnosť poskytuje bezplatný demo účet, kde sa užívateľ môže oboznámiť s funkcionalitami nástroja.

[5] `https://www.jinfonet.com/product/reporting-software/`

[6] `https://www.finereport.com/en/`

JasperReports Library[7]

Populárna open-source knižnica pre tvorbu reportov. Je napísaná v jazyku Java a je ju teda možné integrovať len do Java aplikácii. Možnosť vytvárať s podporou interaktívnych prvkov tabuľky, krížové tabuľky, grafy alebo dashboardy. Podporuje viacero typov dátových zdrojov ako napríklad relačné databázy, XML alebo CSV súbory a tiež mnoho výstupných formátov, ako napríklad PDF, HTML, CSV a XML.

Spoločnosť *JasperSoft*, ktorá je autorom tejto knižnice tiež ponúka kompletný reportovací nástroj *TIBCO JasperReports Server*[8] s podobnými funkcionalitami ako predošlé dva popisované nástroje. Podobne ako *Fine Report*, je možné ho používať samostatne alebo integrovať do webovej aplikácie. Spoločnosť ponúka na tento nástroj aj trial licenciu.

2.4.1 Zhrnutie prehľadu

Nástroj *JReport* je poskytovaný ako doplnok do webových aplikácii a nie ako samostatná služba. Naviac poskytuje nadštandardnú funkcionalitu, ktorá je pre potreby jednoduchých pravidelných tabuľkových reportov nepotrebná. *Fine Report* a *TIBCO JasperReports Server* prinášajú oproti *JReport* výhodu v tom, že je možné ich používať ako samostatný nástroj bez potreby integrácie do iného systému. Spoločnou výhodou spomínaných nástrojov je to, že pre tvorbu reportov nie sú potrebné znalosti z oblasti programovacích jazykov a databáz.

Naproti tomu, použitie knižnice *JasperReports Library* vyžaduje pokročilé znalosti programovania a práce s databázou. Avšak výhodou tejto knižnice je to, že je úplne zdarma a pre spoločnosti, ktoré nepotrebujú veľmi prepracované reporty a zároveň majú svoj tím vývojárov sa stáva veľmi zaujímavou možnosťou.

[7] `https://www.jaspersoft.com/products/jasperreports-library`

[8] `https://www.jaspersoft.com/products/jasperreports-server`

3 Návrh nového systému

V tejto časti práce analyzujem požiadavky zadávateľa na nový informačný systém a následne vytvorím návrh systému. Účelom zavedenia nového systému je odstrániť nedostatky súčasného riešenia a tiež sprehľadniť správu reportov. Požiadavky na systém boli viackrát konzultované so zastupujúcou osobou zo strany zadávateľa. Štruktúra tejto kapitoly je inšpirovaná IEEE 830 šablónou pre dokumenty špecifikácie softvérových požiadaviek (Software Requirements Specification [19]) a ukážkou dokumentu návrhu softvérovej architektúry (Software Architecture Document [20]).

3.1 Opis systému

Nasleduje charakteristika funkcii, ktoré zadávateľ požaduje od nového systému. Jedná sa o prvotnú predstavu o výslednom produkte. Spracovaná bola na základe konzultácii so zadávateľom.

3.1.1 Vízia

Chceme vytvoriť informačný systém, ktorý bude predstavovať centralizované úložisko vygenerovaných reportov. Zamestnanci spoločnosti budú mať na základe oprávnení sprístupnené určité reporty. Do systému budú mať podľa potreby prístup aj externé osoby. Reporty sa budú generovať automaticky a pravidelne v definovanom čase sa následne uložia do centralizovaného úložiska. Reporty si bude môcť užívateľ stiahnuť do svojho zariadenia. Administrátor systému bude mať prehľad o udalostiach v systéme a prípadných chybách.

Pre lepšiu prehľadnosť si určíme kľúčové body ako hlavné požiadavky na výsledný produkt. Tieto požiadavky sa nazývajú aj biznis požiadavky a sú písané na najnižšej úrovni podrobnosti.

BR1 Zabezpečený prístup do systému

BR2 Správa externých užívateľov

BR3 Správa reportov

BR4 Autorizovaný prístup ku reportom

BR5 Zaznamenávanie udalostí v systéme

BR6 Automatické generovanie reportov

3.1.2 Charakteristiky a úlohy užívateľov

So systémom budú pracovať 3 typy užívateľov:

Administrátor

Bude určovať cieľovú skupinu užívateľov, ktorí budú mať prístup ku reportom vzniknutých z danej definície. Ďalšou úlohou administrátora je spravovať registrácie užívateľov mimo spoločnosti.

Administrátor bude mať taktiež prehľad o udalostiach, ktoré v systéme nastanú. Na implementačnej úrovni bude vytvárať definície reportov. To znamená, že bude vytvárať určitú šablónu definujúcu, ktoré dáta z databázy sa pri generovaní reportu budú interpretovať do výstupného reportu a tiež ako. Prihlasovať sa bude pomocou účtu AD. Slovné spojenie „definícia reportu“ bude v takomto význame používané vo zvyšku tejto práce.

Bežný užívateľ

Zamestnanec spoločnosti. Tento užívateľ bude mať dostupné len reporty, ktoré vzniknú z takej definície, u ktorej mu administrátor udelil prístup. Dostupné reporty si tento užívateľ bude môcť stiahnuť do svojho zariadenia. Na vyžiadanie si môže vygenerovať nový report z definície, ku ktorej má prístup. Prihlasovať sa bude taktiež cez AD.

Registrovaný užívateľ

Predstavuje osobu, ktorá nie je zamestnancom spoločnosti, ale zároveň jej bol umožnený prístup do systému. Oproti bežnému užívateľovi sa tento líši len spôsobom prihlasovania. Bude využívať svoju mailovú adresu a heslo. Naviac, bude mať možnosť vyžiadať si vygenerovanie nového hesla a zmeniť si dočasné.

3.1.3 Funkcie systému

Nasleduje výpis funkcii, ktoré požadujeme od výsledného systému. Keďže potrebujeme zaznamenať aj určité funkcie, ktoré musí vykonávať systém bez interakcie užívateľa, rozdelili sme množinu funkcionalít na prípady užitia a systémové činnosti. Tieto funkcie budú následne v kapitole 3.2.1 spracované ako funkčné požiadavky.

Na obrázku 11 je zobrazený diagram prípadov užitia, ktorý priraďuje jednotlivé prípady užitia ku konkrétnym rolám užívateľov. Pre úplnosť pohľadu na výsledný systém obsahuje diagram aj systémové činnosti.

Prípady užitia

UC1 Prihlásenie do systému

UC2 Odhlásenie zo systému

UC3 Zmena prvotného hesla

UC4 Žiadosť o reset hesla

UC5 Registrácia užívateľov

UC6 Evidencia registrovaných užívateľov

UC7 Evidencia definícií reportov

UC8 Inicializácia definícií reportov

UC9 Určenie cieľovej skupiny užívateľov pre definíciu reportu

UC10 Správa definície reportu

UC11 Vytvorenie definície reportu

UC12 Plánovanie generovania reportov

UC13 Evidencia dostupných reportov

UC14 Export reportu

UC15 Manuálne vygenerovanie reportu

UC16 Evidencia logovaných udalostí

UC17 Detail logovanej udalosti

Systémové činnosti

SF1 Vygenerovanie reportu

SF2 Odoslanie reportu cieľovej skupine

SF3 Zaznamenávanie systémových udalostí

3.1.4 Predpoklady a závislosti

Aby systém mohol vykonávať svoju činnosť správne, musia byť splnené nasledovné predpoklady:

AS1 Pre používanie systému musí byť dostupné internetové pripojenie

AS2 Pre funkčné odosielanie mailových správ je potrebná dostupnosť SMTP servera

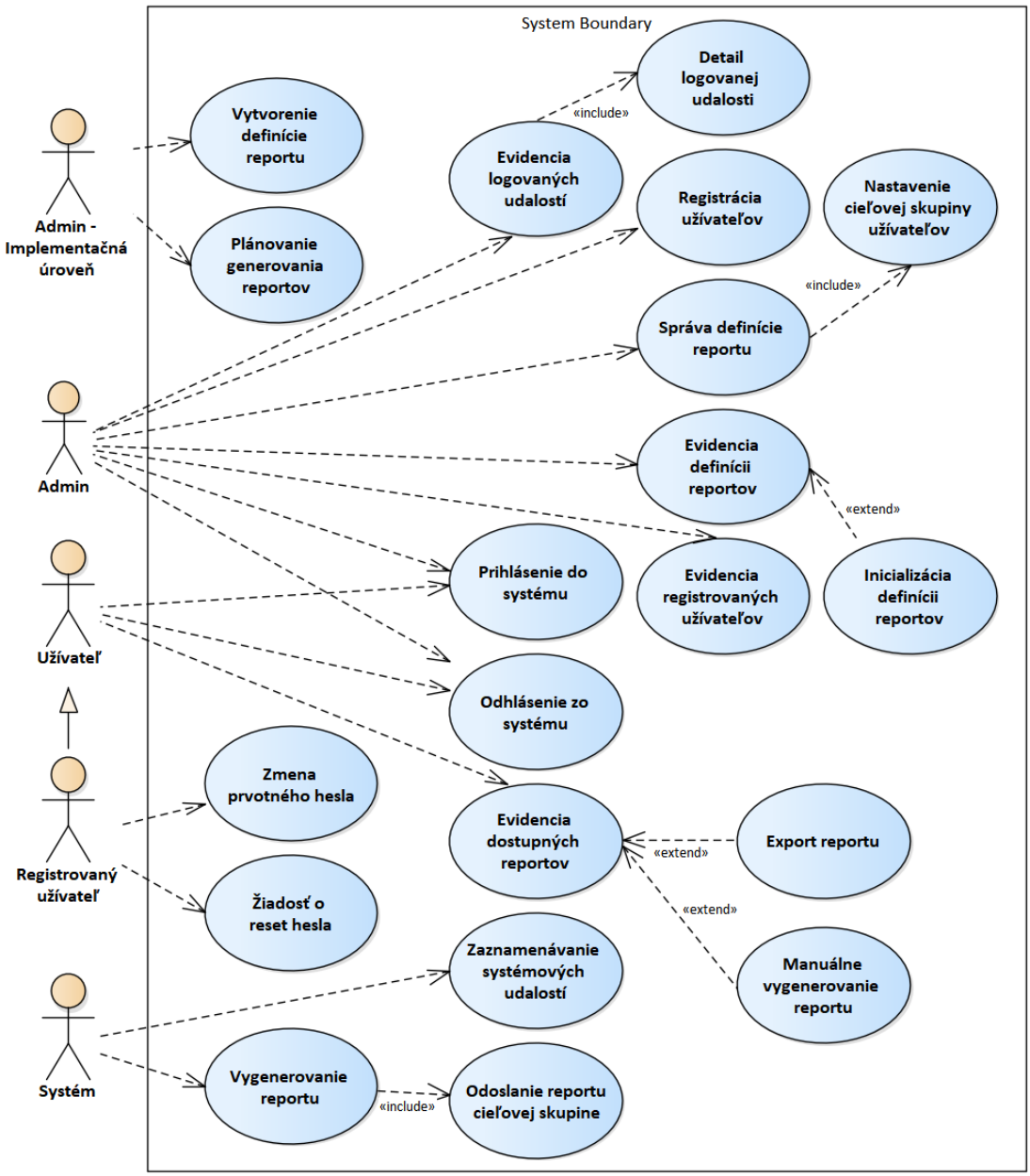
AS3 Pre funkčné generovanie reportov musí byť dostupná databáza s potrebnými dátami

AS4 Pre funkčné prihlasovanie registrovaných užívateľov musí byť dostupná databáza s údajmi o užívateľoch

AS5 Pre funkčné prihlasovanie bežných užívateľov a administrátora musí byť dostupné AD

3.2 Špecifické požiadavky

Doterajší popis systému bol všeobecný a poskytoval nám len základný obraz o výslednom produkte. Táto sekcia sa zameriava na podrobnejšiu špecifikáciu požiadaviek na systém. Tie rozlišujeme na funkčné a kvalitatívne. Funkčné požiadavky popisujú, čo očakávame, že bude systém vykonávať, zatiaľ čo kvalitatívne požiadavky špecifikujú ako by mal systém fungovať. Kvalitatívne požiadavky dopĺňajú funkčné a môžeme medzi ne zaradiť napríklad výkonnosť, spoľahlivosť, použiteľnosť alebo bezpečnosť. Ďalej môžeme špecifikovať obmedzenia, ktoré môžu ovplyvniť návrh systému a tiež rozhrania vyžadované alebo poskytované systémom.



Obr. 11: Diagram prípadov užitia

3.2.1 Funkčné požiadavky

Zoznam funkčných požiadaviek korešponduje so zoznamom funkcií systému uvedených v sekcii 3.1.3. V tejto sekcii ich však zadefinujeme podrobnejšie, určíme, ktoré biznis požiadavky uspokojujú a označíme ich ako funkčné požiadavky.

FR1 Prihlásenie do systému (všetci užívatelia) [UC1, uspokojuje BR1]

FR1.1 Systém bude schopný prihlásiť administrátora a bežného užívateľa automaticky, pomocou účtu AD, pokiaľ sa prihlasuje v rámci firemnej domény.

FR1.2 Systém bude schopný prihlásiť administrátora a bežného užívateľa pomocou mena a hesla účtu AD, pokiaľ sa prihlasuje mimo firemnú doménu.

FR1.3 Systém bude schopný prihlásiť platného registrovaného užívateľa pomocou mailu a hesla.

FR1.4 Systém bude schopný po prihlásení rozlíšiť administrátora a bežného užívateľa na základe role alebo jeho zaradenia do určitej skupiny v AD.

FR2 Odhlásenie zo systému (všetci užívatelia) [UC2, uspokojuje BR1]

FR2.1 Systém bude schopný ukončiť užívateľskú reláciu odhlásením zo systému.

FR3 Zmena prvotného hesla (registrovaný užívateľ) [UC3, uspokojuje BR1]

FR3.1 Systém bude schopný vyžiadať si od registrovaného užívateľa zmenu hesla v prípade, ak sa prihlasuje pomocou dočasného hesla (prvotného heslo alebo heslo vygenerované po jeho resetovaní).

FR3.2 Systém bude schopný overiť, že užívateľove nové heslo obsahuje minimálne jeden veľký, jeden malý a jeden špeciálny znak, jednu číslicu a jeho dĺžka je minimálne 8 znakov.

FR3.3 Systém bude schopný uložiť nové heslo.

FR4 Žiadosť o reset hesla (registrovaný užívateľ) [UC4, uspokojuje BR1]

FR4.1 Systém umožní užívateľovi na základe poskytnutej emailovej adresy odoslať žiadosť o reset hesla.

FR4.2 Systém bude schopný na základe žiadosti užívateľa mu vygenerovať nové náhodné dočasné heslo.

FR4.3 Systém bude schopný odoslať nové heslo užívateľovi na jeho mailovú adresu.

FR5 Registrácia užívateľov (admin) [UC5, uspokojuje BR2]

FR5.1 Systém umožní administrátorovi vytváranie prístupových účtov do systému pre užívateľov na základe emailovej adresy.

FR5.2 Systém bude schopný vygenerovať registrovanému užívateľovi dočasné heslo.

FR5.3 Systém bude schopný odoslať prvotné heslo užívateľovi na zadanú mailovú adresu.

FR5.4 Systém bude schopný detegovať pokus o vytvorenie účtu pre už existujúceho užívateľa.

FR6 Evidencia registrovaných užívateľov (admin) [UC6, uspokojuje BR2]

FR6.1 Systém bude schopný zobraziť administrátorovi všetkých platných užívateľov, ktorých registroval do systému.

FR7 Evidencia definícii reportov (admin) [UC7, uspokojuje BR3]

- **FR7.1** Systém bude schopný zobraziť administrátorovi všetky nadefinované definície reportov.

FR8 Inicializácia definícií reportov (admin) [UC8, uspokojuje BR3]

- **FR8.1** Systém bude schopný detegovať nové definície reportov.
- **FR8.2** Systém bude schopný uložiť informácie o definíciách reportov do databázy.

FR9 Nastavenie cieľovej skupiny užívateľov pre definíciu reportu (admin) [UC9, uspokojuje BR4]

- **FR9.1** Systém bude schopný zobraziť administrátorovi všetkých užívateľov v AD.
- **FR9.2** Systém bude schopný zobraziť administrátorovi všetky skupiny v AD.
- **FR9.3** Systém bude schopný zobraziť administrátorovi všetkých registrovaných užívateľov.
- **FR9.4** Systém umožní administrátorovi nastaviť užívateľom a skupinám prístup ku reportom podľa definície reportu.

FR10 Správa definície reportu (admin) [UC10, uspokojuje BR3]

- **FR10.1** Systém umožní administrátorovi povoliť/pozastaviť generovanie reportov z danej definície.
- **FR10.2** Systém umožní zapnúť/vypnúť odosielanie informatívnych mailov cieľovej skupine užívateľov.

FR11 Vytvorenie definície nového reportu (admin – implementačná úroveň) [UC11, uspokojuje BR6]

- **FR11.1** Systém bude na implementačnej úrovni poskytovať vhodné rozhranie, ktorého implementáciou administrátor vytvorí definíciu reportu.

FR12 Plánovanie generovania reportov (admin – implementačná úroveň) [UC12, uspokojuje BR6]

- **FR12.1** Systém bude na implementačnej úrovni poskytovať spôsob nastavenia frekvencie generovania reportu.

FR13 Evidencia dostupných reportov (užívateľ) [UC13, uspokojuje BR3]

- **FR13.1** Systém bude schopný zobraziť užívateľovi reporty, ku ktorým má prístup.
- **FR13.2** Systém bude schopný vyhľadávať v dostupných reportoch u užívateľa podľa názvu reportu alebo názvu definície, z ktorej daný report vznikol.
- **FR13.3** Systém bude schopný filtrovať zobrazené reporty podľa dátumu.

FR14 Export reportu (užívateľ) [UC14, uspokojuje BR3]

FR14.1 Systém umožní registrovanému a bežnému užívateľovi stiahnuť report, ku ktorému má prístup.

FR15 Manuálne vygenerovanie reportu (užívateľ)[UC15, uspokojuje BR6]

FR15.1 Systém bude schopný spustiť okamžité generovanie reportu na podnet užívateľa.

FR16 Evidencia logovaných udalostí (admin) [UC16, uspokojuje BR5]

FR16.1 Systém bude schopný zobraziť administrátorovi zaznamenané udalosti v systéme.

FR16.2 Systém bude schopný filtrovať logované udalosti podľa typu.

FR16.3 Systém bude schopný filtrovať logované udalosti podľa dátumu.

FR16.4 Systém bude schopný filtrovať logované udalosti podľa textového reťazca obsiahnutého v správe udalosti.

FR17 Detail logovanej udalosti (admin) [UC17, uspokojuje BR5]

FR17.1 Systém bude schopný zobraziť administrátorovi detail vybranej zaznamenanej udalosti v systéme.

FR18 Vygenerovanie reportu (systém) [SF1, uspokojuje BR6]

FR18.1 Systém bude schopný v stanovenom čase vygenerovať report podľa nastavení jeho definície.

FR18.2 Systém bude schopný uložiť report na centralizované úložisko.

FR18.3 Systém bude logovať vygenerovanie reportu.

FR18.4 Systém bude schopný odoslať administrátorovi mail v prípade chyby pri generovaní reportu.

FR19 Odoslanie reportu cieľovej skupine (systém) [SF2, uspokojuje BR4]

FR19.1 Systém bude schopný sprístupniť určeným užívateľom vygenerovaný report.

FR19.2 Systém bude schopný odoslať informatívny mail s odkazom na report v systéme cieľovej skupine užívateľov.

FR20 Zaznamenávanie systémových udalostí [SF3, uspokojuje BR5]

FR20.1 Systém bude schopný evidovať systémové udalosti do databázy.

FR20.2 Systém bude schopný evidovať:

- vygenerovanie reportu,
- žiadosť o reset hesla,

- registrovanie užívateľa,
- inicializáciu definícii reportov,
- manuálne vygenerovanie reportu,
- chybové udalosti vyvolané v kóde.

3.2.2 Kvalitatívne požiadavky

Výkonnosť

QR1 Systém bude schopný pracovať s odozvou menšou ako 2 sekundy

QR2 Systém bude schopný v jednom čase obslúžiť minimálne 30 používateľov

Bezpečnosť

Systém bude spracovávať firemné údaje, preto je nutné myslieť aj na zabezpečenie systému. Keďže každý zamestnanec má tiež účet v Active Directory firemnej domény, pre prihlasovanie so systému bude využitá existencia týchto účtov. Zadávateľ tiež požadoval, aby do systému mali prístup aj osoby bez AD účtu a preto bude v systéme možnosť vytvárať užívateľom účty pre prístup do systému.

Užívateľ si môže zobraziť len také reporty, ktoré vznikli z definície, ktorá je mu prístupná. Meniť prístupy ku reportom môže len administrátor. Taktiež len administrátor môže registrovať nového užívateľa.

Systém bude rozlišovať role (ADMIN alebo USER) u registrovaných užívateľov pomocou špeciálneho príznaku, ktorý bude mať každý užívateľ. U užívateľov v AD sa administrátor bude odlišovať tým, že bude patriť do doménovej skupiny „GLOVIS\GCZ_IT“.

3.2.3 Dizajnové a implementačné obmedzenia

Na základe požiadavku zadávateľa bude systém implementovaný ako webová aplikácia vo frameworku ASP.NET pomocou Web Forms. Pre dizajn budú použité logá a farby vizuálneho štýlu spoločnosti *Hyundai Glovis Czech Republic s.r.o.*. Počas implementácie budú zohľadnené vopred dohodnuté menné konvencie. Pre objektovo-relačné mapovanie dát z MySQL databázy bude využitý Entity Framework 6.

Knižnice tretích strán

- **ClosedXML** - Slúži na vytváranie .xlsx súborov priamo v aplikácii. Bude využité pri generovaní tabuľkových reportov.
- **FluentScheduler** - Jednoduchý plánovač úloh so širokou možnosťou nastavení plánovania. Bude využité pre plánovanie generovania reportov.
- **Log4Net** - Nástroj na logovanie udalostí v systéme s možnosťou ukladania záznamov priamo do databázy.

Vývojové nástroje

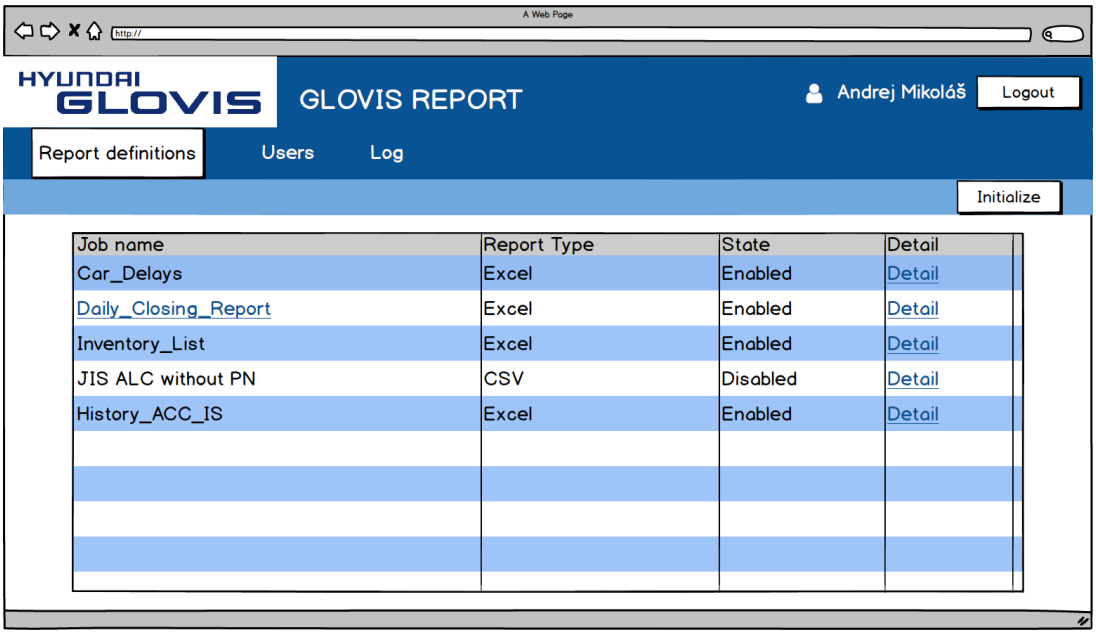
Za účelom vývoja systému bude poskytnutý VPN prístup do firemnej siete a následne na vzdialenú pracovnú plochu firemného serveru. Na tomto serveri bude prebiehať vývoj pomocou vývojového prostredia Microsoft Visual Studio 2015 Community Edition. Pre verziovanie zdrojového kódu bude využitý verziovací systém Subversion, ktorý je tiež hostovaný na firemnom serveri.

3.2.4 Rozhrania

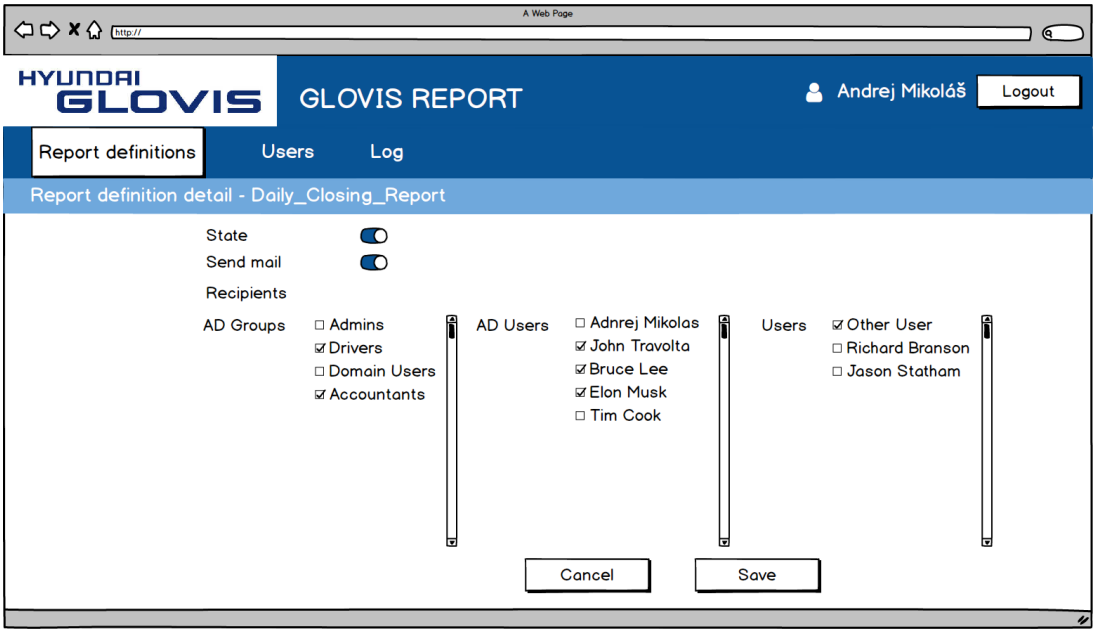
Užívateľské rozhranie

Funkcionalita a rozsah systému nie sú veľké, preto užívateľské rozhranie bude zamerané na jednoduchosť a účelnosť. Keďže aplikácia bude prevažne používaná skrz webový prehliadač Google Chrome, pri vytváraní UI bude braný ohľad na kompatibilitu použitých vizuálnych komponent s týmto prehliadačom.

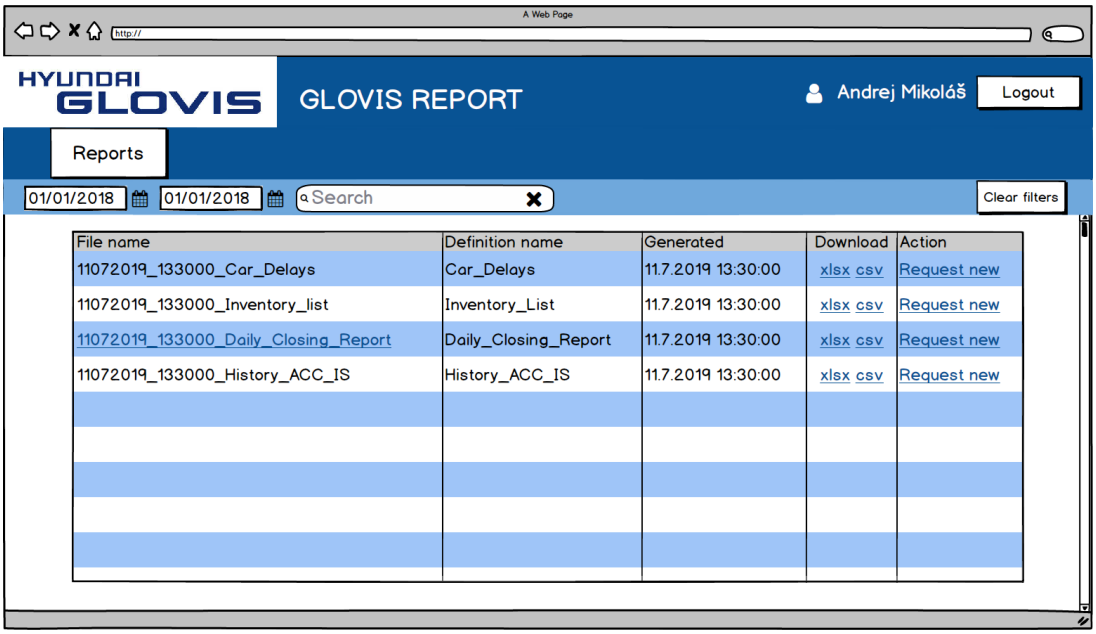
Na obrázkoch 12, 13 a 14 sú ukážky návrhov užívateľského rozhrania systému. Farebne je rozhranie ladené do odtieňov modrej farby, ktorá je pre spoločnosť príznačná. Pre oba hlavné typy užívateľov bude spoločná hlavná lišta, obsahujúca logo spoločnosti, názov aplikácie, meno prihláseného užívateľa a tlačidlo pre odhlásenie zo systému. Pod hlavnou lištou sa nachádza navigačná lišta, ktorej položky sa budú líšiť medzi administrátorskou a užívateľskou verziou. Nasleduje pracovná lišta, rozlíšená svetlejšou farbou. Bude obsahovať ďalšie akcie, prípadne prvky, ktoré budú filtrovať obsah zobrazený v pracovnej ploche. Tá tvorí zvyšnú oblasť na obrazovke pod lištami.



Obr. 12: Návrh UI - Administrátor - zoznam definícii reportov



Obr. 13: Návrh UI - Administrátor - správa definície reportu



Obr. 14: Návrh UI - Užívateľ - zoznam dostupných reportov

Hardvérové rozhrania

Tento systém nevyžaduje žiadne zvláštne hardvérové rozhrania.

Softvérové rozhrania

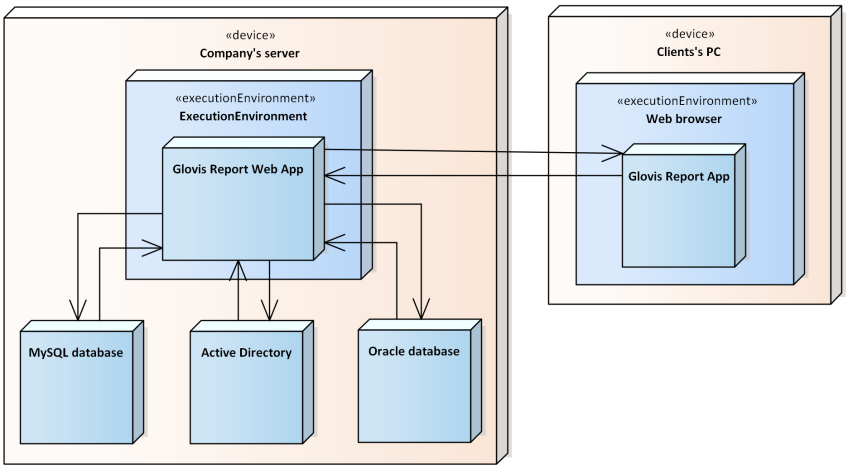
Ako už bolo spomínané v sekcii 3.2.2, pre prihlasovanie zamestnancov do systému sa bude využívať adresárová služba Active Directory. V tejto službe je každý užívateľ zaradený do jednej alebo viacerých doménových skupín. Informácie o užívateľoch a skupinách v tejto službe sa budú využívať pre kontrolu prístupu užívateľov k určitým reportom.

3.3 Architektúra systému

V tejto časti práce bude podrobnejšie navrhnutý systém z rôznych pohľadov. Určíme, ako bude systém nasadený, rozvrhneme systém do logických komponentov, zadefinujeme hlavné doménové objekty a ich vzťahy a tiež analyzujeme prípady užitia, ktoré vyžadujú interakciu užívateľa.

3.3.1 Architektonické ciele a obmedzenia

Webová aplikácia bude nasadená na serveri firemnej infraštruktúry a prístupná cez webový prehliadač klienta. Primárne bude aplikácia používaná cez prehliadač Google Chrome. Samotná aplikácia bude komunikovať s databázami typu Oracle a MySQL. Aplikácia bude mať prístup aj do adresárovej služby Active Directory spoločnosti. Diagram nasadenia je zachytený na obrázku 15.



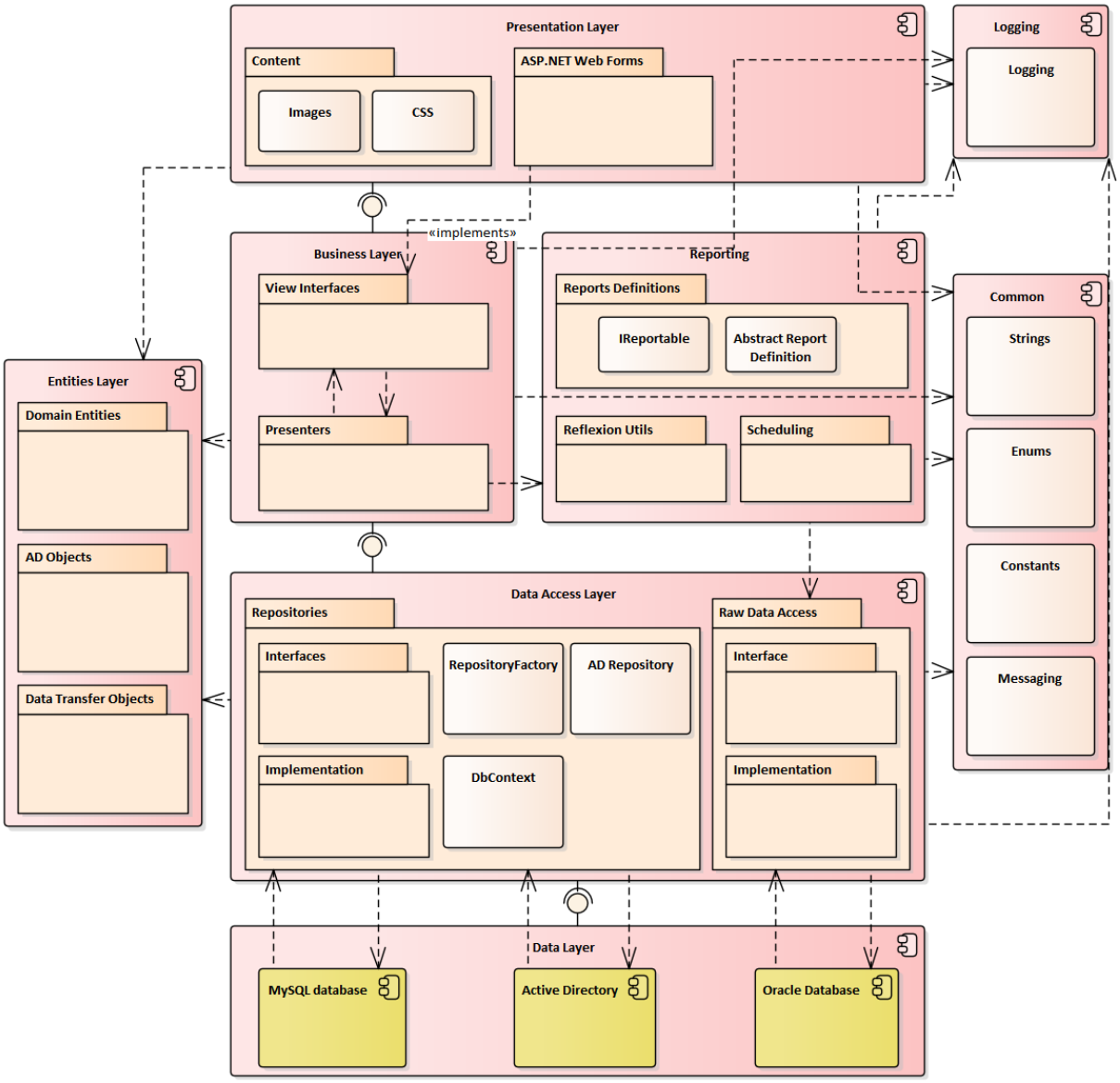
Obr. 15: Diagram nasadenia aplikácie

Dátová perzistencia

Pre ukladanie informácii o užívateľoch, logovaných udalostiach, vzniknutých reportoch a ich definíciách bude slúžiť databáza typu MySQL. Táto databáza je hostovaná na serveri firemnej infraštruktúry. Systém bude tiež komunikovať s databázou typu Oracle, ktorá sa v súčasnosti využíva internými systémami spoločnosti pre ukladanie dát. Z tejto databázy bude systém získavať dáta a generovať reporty.

3.3.2 Logický pohľad

Na obrázku 16 je zobrazený komponentný diagram systému. Takýto typ diagramu zobrazuje aké komponenty tvoria systém, ako sú vzájomne prepojené a tiež ako komunikujú. Komponent predstavuje zapuzdrenú modulárnu jednotku, ktorú je možné samostatne zameniť alebo preniesť do iného systému. Pre implementáciu webovej aplikácie vo frameworku ASP.NET WebForms bude použitý architektonický vzor Model View Presenter, presnejšie Supervising Controller.



Obr. 16: Diagram komponent

Prezentačná vrstva

Prezentačnú vrstvu tvoria stránky ASP.NET WebForms. Tie sa skladajú z `.aspx` súboru pre užívateľské rozhranie a `.cs` súboru prezentačnej logiky. Pre každú stránku existuje v súlade

s MVP rozhranie umiestnené v biznis vrstve, ktoré stránka implementuje. Nachádzajú sa tu taktiež obrázky, mailové šablóny a súbory definujúce štýly.

Biznis vrstva (výpočtová)

Tu sa nachádzajú triedy typu Presenter, jedna pre každú stránku. Táto vrstva zabezpečuje komunikáciu medzi prezentačnou vrstvou a vrstvou pre prístup k dátam. Spolupracuje tiež s reportovacím systémom a spoločnou vrstvou.

Reporting

Kód pre reportovací systém je vyčlenený do osobitnej komponenty, pre prípadnú potrebu prenositeľnosti medzi systémami. Komponent obsahuje triedy definujúce reporty, pomocné triedy pre reflexiu a plánovač úloh. Vrstva využíva podporné služby vrstvy Common a komunikuje s rozhraniami vrstvy pre prístup k dátam.

To, ako sa bude výsledný report generovať, definuje na to určená trieda - definícia reportu. Pre každý typ reportu existuje jedna takáto trieda. Definícia reportu spolu s jej nadradenými triedami obsahuje všetko potrebné pre správne generovanie reportov.

Vrstva prístupu k dátam

Systém bude pristupovať ku dvom typom databázy - MySQL a Oracle. Pre prístup ku MySQL databáze bude použitý návrhový vzor *Repository* v spolupráci so vzorom *Abstract Factory*. Implementovaný repozitár bude využívať Entity Framework 6. Pre získavanie dát z Active Directory bude dostupná pomocná trieda. Dáta z Oracle databázy bude vrstva poskytovať vo forme dátových setov na základe poskytnutého SQL dotazu. Aj u tejto databázy bude využitý vzor *Abstract Factory*. Pre obe databázy bude vrstva poskytovať osobitné rozhranie.

Dátová vrstva

Dátová vrstva predstavuje všetky dátové úložiská, s ktorými systém pracuje. Každé z nich je umiestnené na inom serveri firemnej infraštruktúry a každé vyžaduje odlišný prístup pre prácu s ním. Patria sem tieto úložiská:

- MySQL databáza: úložisko pre informácie o reportoch, registrovaných užívateľoch a logovaných udalostiach (čítanie a zápis),
- Oracle databáza: zdroj údajov, z ktorých systém generuje reporty (len čítanie),
- Active Directory: adresárový systém, využívaný pre prihlasovanie užívateľov v rámci firemnej domény (len čítanie).

Spoločná vrstva

Táto vrstva zabezpečuje podpornú funkcionalitu pre väčšinu vrstiev. Zahŕňa vlastného mailového klienta, konštanty, lokalizované textové reťazce a vymenované typy.

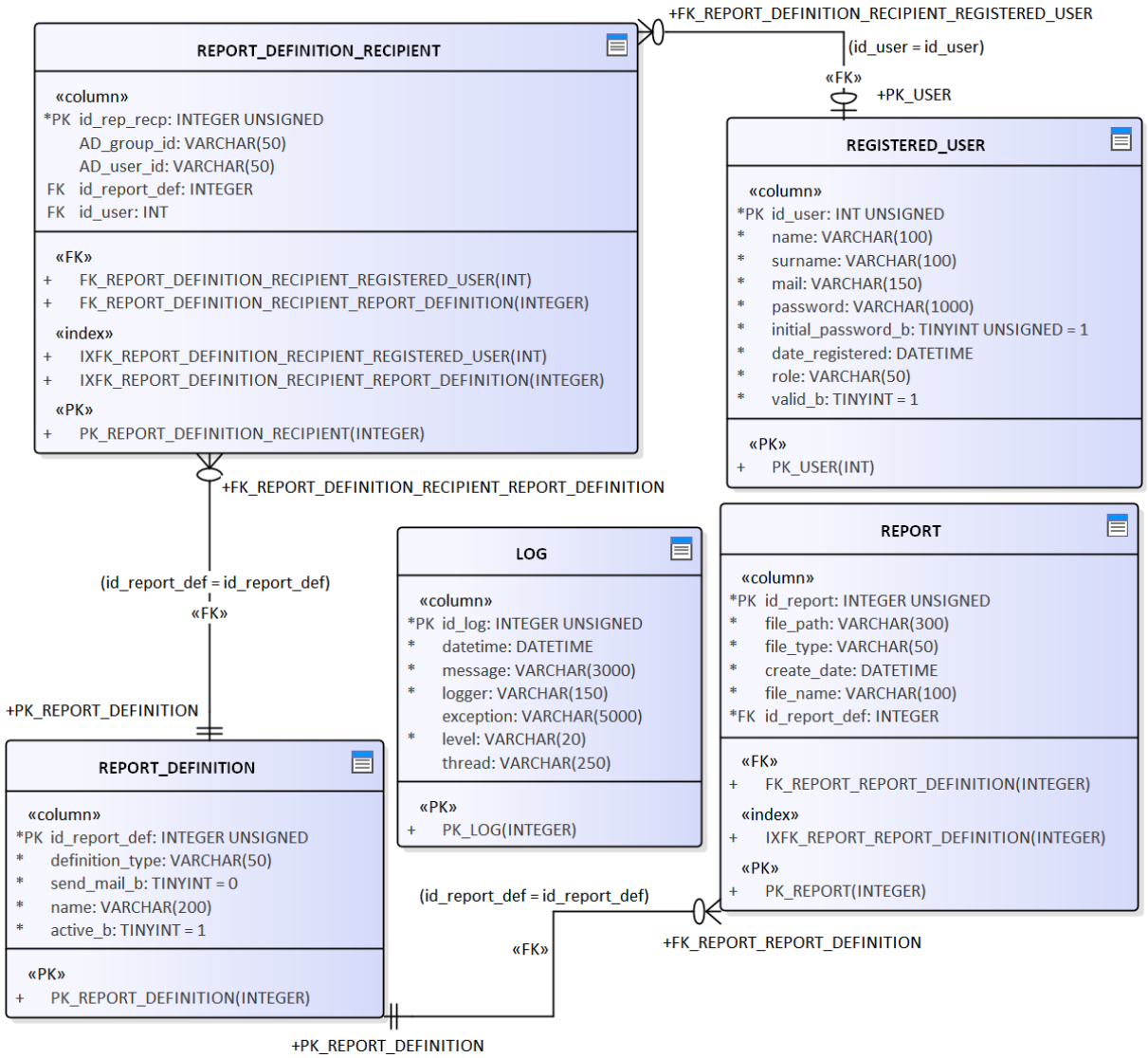
Vrstva entít

Obsahuje dátové objekty, ktoré predstavujú hlavné doménové objekty v systéme. V niektorých prípadoch bude potrebné z dátového zdroja získavať dáta v určitých väzbách, prípadne len ich niektoré atribúty. Pre tento účel bude táto vrstva obsahovať aj tzv. Data Transfer Objects. Pre prácu s dátami o užívateľoch a skupinách v AD budú existovať zjednodušené objekty.

Logovanie

Pre využívanie pokročilejšieho logovania budú mať všetky komponenty, kde sa to vyžaduje, referencie na knižnicu *Log4Net*, ktorá bola spomínaná v sekcii 3.2.3.

3.3.3 Doménový pohľad



Obr. 17: Diagram logického modelu

Reporty

Entita `REPORT_DEFINITION` predstavuje definíciu reportu v databázovom modeli. Obsahuje unikátny identifikátor, ktorým je previazaná na triedu definície reportu, ale aj názov samotnej definície a typ výstupného súboru. Pri inicializácii definícií sa pomocou reflexie prejdú všetky triedy, implementujúce rozhranie `IReportable`, získajú sa ich identifikátory a porovnajú sa so

záznamami v `REPORT_DEFINITION`. Ak systém deteguje novú definíciu, vytvorí sa v tejto tabuľke nový záznam s daným identifikátorom.

Entita `REPORT_DEFINITION_RECIPIENT` predstavuje osobu alebo skupinu v AD, alebo registrovaného užívateľa, ktorý je autorizovaný prehliadať reporty vzniknuté z danej definície reportu. Každý záznam v tejto tabuľke vždy predstavuje len jeden typ prijímateľa. V prípade užívateľa alebo skupiny v AD bude záznam obsahovať identifikátor do AD a v prípade registrovaného užívateľa referenciu na záznam v tabuľke `REGISTERED_USER`. Medzi entitami `REPORT_DEFINITION` a `REPORT_DEFINITION_RECIPIENT` bude vzťah 1:N.

Entita `REPORT` predstavuje reporty, ktoré sa budú generovať v stanovených intervaloch. Záznam v `REPORT` vždy vychádza z `REPORT_DEFINITION` a je medzi nimi väzba 1:N. Každý záznam obsahuje cestu ku vygenerovanému súboru a dátum a čas vzniku.

Tabuľka 1: Dátový slovník pre entitu REPORT_DEFINITION

Atribút	**Typ**	**Dĺžka**	**IO**	**Null?**	**Default**	**Popis**
id_report_def	LONG	-	1	Nie	-	identifikátor mapujúci záznam na triedu definície
name	VARCHAR	200		Nie	-	názov definície reportu
definition_type	VARCHAR	50		Nie	-	typ výstupného súboru
active_b	TINYINT	-	2	Nie	1	príznak indikujúci, či je definícia aktívna
send_mail_b	TINYINT	-	2	Nie	0	príznak indikujúci, či sa má odosielať mail po vygenerovaní reportu

Tabuľka 2: Dátový slovník pre entitu REPORT

Atribút	**Typ**	**Dĺžka**	**IO**	**Null?**	**Default**	**Popis**
id_report	LONG	-	1	Nie	-	identifikátor reportu
file_path	VARCHAR	300		Nie	-	cesta k súboru na FileServeri
file_type	VARCHAR	50		Nie	-	typ súboru reportu
create_date	DATETIME	-		Nie	SYSDATE	dátum a čas vygenerovania
file_name	VARCHAR	100		Nie	-	názov súboru
id_report_def	INTEGER	-		Nie	-	cudzí kľúč do tabuľky REPORT_DEFINITION odkazujúci na definíciu reportu, z ktorej vznikol
uid	VARCHAR	50	4	Nie	-	identifikátor typu GUID

Tabuľka 3: Dátový slovník pre entitu REPORT_DEFINITION_RECIPIENT

Atribút	**Typ**	**Dĺžka**	**IO**	**Null?**	**Default**	**Popis**
id_rep_recp	LONG	-	1	Nie	-	identifikátor prijímateľa reportu
AD_group_id	VARCHAR	50	3	Ano	-	identifikátor skupiny užívateľov v Active Directory
AD_user_id	VARCHAR	50	3	Ano	-	identifikátor užívateľa v Active Directory
user_id	INTEGER	-	3	Ano	-	cudzí kľúč do tabuľky REGISTERED_USER
id_report_def	INTEGER	-		Nie	-	cudzí kľúč do tabuľky REPORT_DEFINITION odkazujúci na definíciu reportu, ktorej priraďuje prijímateľov

Integritné obmedzenia:

1. Primárny unikátny kľúč
2. Nadobúda iba hodnoty: {0, 1}
3. V jednom zázname je vždy vyplnený iba jeden z atribútov
4. Unikátna hodnota

Registrovaní užívatelia

Entita `REGISTERED_USER` predstavuje registrovaných užívateľov. Záznamy vytvára administrátor na základe mailovej adresy, mena a priezviska. Heslo sa bude uchovávať vo forme jeho hash hodnoty. Záznamy v tejto tabuľke sa nebudú vymazávať, ale bude sa len meniť hodnota atribútu `VALID_B`.

Tabuľka 4: Dátový slovník pre entitu REGISTERED_USER

Atribút	**Typ**	**Dĺžka**	**IO**	**Null?**	**Default**	**Popis**
id_user	LONG	-	1	Nie	-	identifikátor registrovaného užívateľa
name	VARCHAR	100		Nie	-	meno užívateľa
surname	VARCHAR	100		Nie	-	priezvisko užívateľa
mail	VARCHAR	150	2	Nie	-	mail užívateľa
password	VARCHAR	1000	3	Nie	-	heslo užívateľa
initial_password_b	TINYINT	1	4	Nie	1	príznak, či je heslo užívateľa len dočasné
date_registered	DATETIME	-		Nie	SYSDATE	dátum a čas registrácie
role	VARCHAR	50	5	Nie	USER	rola užívateľa
valid_b	TINYINT	1	4	Nie	1	príznak, či sa jedná o platný záznam
uid	VARCHAR	50	6	Nie	-	identifikátor typu GUID

Integritné obmedzenia:

1. Primárny unikátny kľúč
2. Textový reťazec v platnom formáte mailovej adresy
3. Len hash hodnota hesla
4. Nadobúda iba hodnoty: {0, 1}
5. Nadobúda iba hodnoty 'ADMIN' alebo 'USER'
6. Unikátna hodnota

Logovanie

Entita `LOG` predstavuje záznam o udalosti vyvolanej v systéme. Záznamy vytvára systém pomocou nástroja *Log4Net*.

Tabuľka 5: Dátový slovník pre entitu LOG

Atribút	Typ	Dĺžka	IO	Null?	Default	Popis
id_log	LONG	-	1	Nie	-	identifikátor udalosti
datetime	DATETIME	-		Nie	SYSDATE	dátum a čas vzniku udalosti
message	VARCHAR	3000		Nie	-	text udalosti
logger	VARCHAR	150		Nie	-	názov komponenty, ktorá vyvolala udalosť
exception	VARCHAR	5000		Áno	-	obsah vyvolanej výnimky
level	VARCHAR	20	2	Nie	-	úroveň závažnosti
thread	VARCHAR	250		Nie	-	vlákno

Integritné obmedzenia:

1. Primárny unikátny kľúč
2. Nadobúda hodnoty z {FATAL, ERROR, WARN, INFO, DEBUG}

3.3.4 Analýza prípadov užitia

V tejto sekcii podrobne rozpíšeme jednotlivé prípady užitia, ktoré vyžadujú interakciu užívateľa cez užívateľské rozhranie. U každého prípadu užitia bude mimo iné uvedené aj označenie funkčných požiadaviek zo sekcie 3.2.1, ktoré uspokojujú.

UC0 Spoločná funkcionalita

Aktéri: administrátor, bežný užívateľ, registrovaný užívateľ
Použité tabuľky: `REGISTERED_USER`
Hlavná lišta:

Obr. 18: UC0 - hlavná lišta

Logo [E01] (obrázok)

- Zobrazenie: logo *Hyundai Glovis s.r.o.*

Meno užívateľa [E02] (text)

- Zobrazenie: meno aktuálne prihláseného užívateľa získané z užívateľskej relácie

Odhlásiť (UC2 Odhlásenie zo systému) [E03] (tlačidlo)

- Zobrazenie: tlačidlo s lokalizovaným textom „*Odhlásiť*“
- Ovládanie:
  - kliknutím sa ukončí užívateľská relácia [FR2.1]
  - dôjde k presmerovaniu užívateľa na obrazovku UC1 Prihlásenie do systému

Lišta administrátora:

Obr. 19: UC0 - lišta administrátora

Menu [O02] (navigačné menu)

- Zobrazenie: horizontálne menu s položkami „*Definície reportov*“ , „*Užívatelia*“ , „*Log*“
- Ovládanie:
  - Kliknutím na položky dôjde k presmerovaniu na konkrétne UC
    - Definície reportov [E01] -> UC7 Evidencia definícií reportov
    - Užívatelia [E02] -> UC6 Evidencia registrovaných užívateľov
    - Log [E03] -> UC16 Evidencia logovaných udalostí

Lišta užívateľa:

Obr. 20: UC0 - lišta užívateľa

Menu [O02] (navigačné menu)

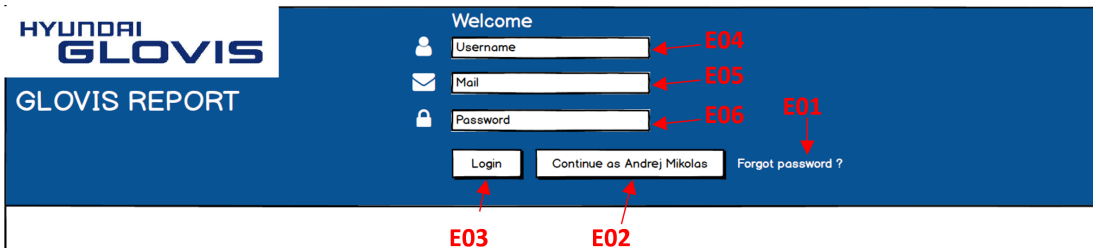
- Zobrazenie: horizontálne menu s položkou „*Reporty*“
- Ovládanie:
  - Kliknutím na položku dôjde k presmerovaniu na konkrétne UC
    - Reporty [E01] -> UC13 Evidencia dostupných reportov

UC1 Prihlásenie do systému

Aktéri: administrátor, bežný užívateľ, registrovaný užívateľ
Predpoklady:

- aktér má vytvorený účet v AD firemnej domény alebo je registrovaný v systéme
- aktér momentálne nie je prihlásený v systéme

Spúšťač: užívateľ chce vstúpiť do systému (zahájiť reláciu)
Použité tabuľky: `REGISTERED_USER`
Okno sa spustí pri zobrazení domovskej obrazovky systému. Pri načítaní okna sa systém pokúsi získať údaje o aktuálne prihlásenom užívateľovi v AD.
Kompozícia užívateľského rozhrania:



Obr. 21: UC1 - kompozícia užívateľského rozhrania

Zabudnuté heslo? [E01] (tlačidlo)

- Zobrazenie: lokalizovaný text „*Zabudnuté heslo?*“
- Ovládanie: kliknutím dôjde k presmerovaniu na UC4 Žiadosť o reset hesla

Pokračovať ako „. . . “ [E02] (tlačidlo)

- Zobrazenie:

  - tlačidlo je zobrazené iba v prípade, že systém deteguje AD užívateľa prihláseného na danom zariadení
  - zobrazuje sa ako tlačidlo s lokalizovaným textom „*Pokračovať ako*“ + meno a priezvisko užívateľa získaného z AD
- Ovládanie:
  - vytvorí sa nová užívateľská relácia v systéme s užívateľom získaným z AD [FR1.1]
  - na základe role užívateľa je presmerovaný na UC7 Evidencia definícií reportov (ADMIN) alebo na UC13 Evidencia dostupných reportov (USER) [FR1.4]

Prihlásiť [E03] (tlačidlo)

- Zobrazenie: tlačidlo s lokalizovaným textom „*Prihlásiť*“
- Ovládanie:
  - overí sa, že pole E06 je vyplnené
    - ak nie je vyplnené, zobrazí sa varovná hláška
  - v prípade, že je vyplnené pole E04, systém spustí prihlasovanie doménového užívateľa pomocou užívateľského mena a hesla
  - systém overí zadané prihlasovacie údaje voči AD a v prípade úspešnej validácie je užívateľ prihlásený a jeho údaje sú uložené do užívateľskej relácie [FR1.2]
    - v prípade nesprávneho hesla alebo mena sa zobrazí varovná hláška
  - v prípade, že je vyplnené pole E05, systém spustí prihlasovanie registrovaného užívateľa
  - systém na základe poľa E05 vyhľadá užívateľa v tabuľke `REGISTERED_USER` podľa `REGISTERED_USER.MAIL`
    - ak sa užívateľ nenájde zobrazí sa hláška o neexistujúcom užívateľovi
  - následne sa vypočíta hash hodnota z poľa E05 a porovná sa s hash hodnotou `REGISTERED_USER.PASSWORD`
    - v prípade nezhody sa zobrazí hláška o nesprávnom hesle
  - v prípade hodnoty parametru `REGISTERED_USER.INITIAL_PASSWORD_B = 1` dôjde k presmerovaniu na obrazovku z UC3 Zmena prvotného hesla [FR3.1]
  - v opačnom prípade sa vytvorí nová užívateľská relácia v systéme [FR1.3]
  - na základe role užívateľa je presmerovaný na UC7 Evidencia definícií reportov (ADMIN) alebo na UC13 Evidencia dostupných reportov (USER)

Užívateľské meno [E04] (editovateľné pole)

- Zobrazenie: editovateľné pole s nápoveďou s lokalizovaným textom „*Užívateľské meno*“

Mail [E05] (editovateľné pole)

- Zobrazenie: editovateľné pole s nápoveďou s lokalizovaným textom „*Mail*“

Heslo [E06] (editovateľné pole)

- Zobrazenie: editovateľné pole s nápoveďou s lokalizovaným textom „*Heslo*“
- Ovládanie: zadávaný text sa zobrazuje vo forme znakov ’*’

UC3 Zmena prvotného hesla

Aktéri: registrovaný užívateľ
Predpoklady: aktér má dočasné heslo `(REGISTERED_USER.INITIAL_PASSWORD_B=1)`
Spúšťač: užívateľ sa prihlasuje do systému prvýkrát po registrácii alebo po zmene hesla
Použité tabuľky: `REGISTERED_USER`
Do okna je možné sa dostať po úspešnom prihlásení registrovaného užívateľa do systému.
Kompozícia užívateľského rozhrania:

Obr. 22: UC3 - kompozícia užívateľského rozhrania

Nové heslo [E01] (editovateľné pole)

- Zobrazenie: editovateľné pole s nápoveďou s lokalizovaným textom „*Nové heslo*“
- Ovládanie: zadávaný text sa zobrazuje vo forme znakov ’*’

Potvrdiť heslo [E02] (editovateľné pole)

- Zobrazenie: editovateľné pole s nápoveďou s lokalizovaným textom „*Potvrďte nové heslo*“
- Ovládanie: zadávaný text sa zobrazuje vo forme znakov ’*’

OK [E03] (tlačidlo)

- Zobrazenie: tlačidlo s textom „*OK*“
- Ovládanie:
  - kliknutím sa overí, že polia E01 a E02 sú vyplnené a zároveň, že tieto polia obsahujú rovnaký text
    - ak pole nie je vyplnené, zobrazí sa varovná hláška
    - ak hodnoty nesúhlasia, zobrazí sa hláška o nezhode hesiel
  - následne sa skontroluje, či zadaný text (@newPass) v poli E01 a E02 obsahujú minimálne 8 znakov, 1 malý, 1 veľký a 1 špeciálny znak a 1 číslicu [FR3.2]
    - ak nie, zobrazí sa hláška o slabej kvalite hesla

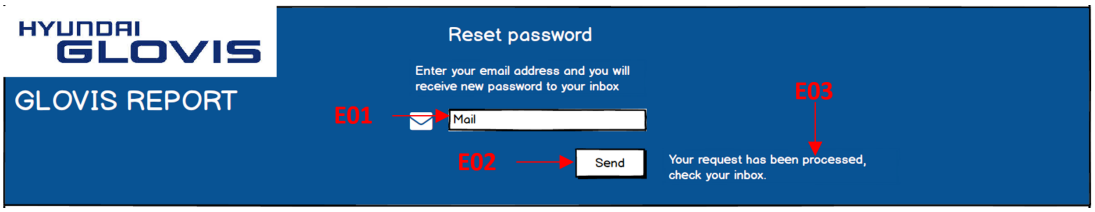
– systém následne aktualizuje heslo u daného užívateľa a heslo označí ako trvalé (`REGISTERED_USER.PASSWORD = hash(@newPass), REGISTERED_USER.INITIAL_PASSWORD_B = '0'`) [FR3.3]
– systém zaeviduje zmenu hesla do logu [FR20.1]

UC4 Žiadosť o reset hesla

Aktéri: registrovaný užívateľ
Predpoklady:

- aktér je registrovaný v systéme
- aktér nie je momentálne prihlásený v systéme

Spúšťač: užívateľ zabudol svoje aktuálne heslo a chce si systémom nechať vygenerovať nové
Použité tabuľky: `REGISTERED_USER`
Kompozícia užívateľského rozhrania:



Obr. 23: UC4 - kompozícia užívateľského rozhrania

Mail [E01] (editovateľné pole)

- Zobrazenie: editovateľné pole s nápoveďou s lokalizovaným textom „*Mail*“

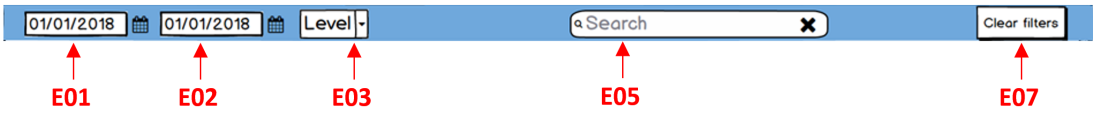
Odoslať [E02] (tlačidlo)

- Zobrazenie: tlačidlo s lokalizovaným textom „*Odoslať*“
- Ovládanie:
  – kliknutím sa overí, že pole E01 je vyplnené
    * ak pole nie je vyplnené, zobrazí sa varovná hláška
  – systém vyhľadá užívateľa s danou mailovou adresou (`REGISTERED_USER.MAIL=E01`) [FR4.1]
    * ak sa užívateľ nenájde, zobrazí sa hláška o neexistujúcom užívateľovi
  – systém vygeneruje nový náhodný 8-miestny reťazec čísel a znakov abecedy (@newPass) a pomocou komponenty pre odosielanie mailov vytvorí a odošle správu s novým heslom (@newPass) na mailovú adresu užívateľa (`REGISTERED_USER.MAIL`) [FR4.2, FR4.3]

- systém aktualizuje heslo u daného užívateľa (`REGISTERED_USER.PASSWORD=hash (@newPass)`) a označí ho ako dočasné (`REGISTERED_USER.INITIAL_PASSWORD_B=1`)
- zobrazí sa E03 s lokalizovaným textom „*Vaša žiadosť bola spracovaná, skontrolujte svoju mailovú schránku*“
- systém zaeviduje žiadosť o reset hesla do logu [FR20.1]

UC16 Evidencia logovaných udalostí

Aktéri: administrátor
Predpoklady: aktér je prihlásený v systéme
Spúšťač: administrátor chce zobraziť zaznamenané udalosti v systéme
Použité tabuľky: `LOG`
Lišta filtrov:



Obr. 24: UC16 - lišta filtrov

Dátum od, dátum do [E01, E02] (komponent pre výber dátumu)

- Zobrazenie: predvolene je zobrazený aktuálny dátum
- Ovládanie:
  - kliknutím sa otvorí komponent pre výber dátumu z kalendára
  - potvrdením nového dátumu (`@dateFrom` alebo `@dateTo`) sa rozšíri filtračná podmienka o zobrazenie len logovaných udalostí, ktoré majú dátum vygenerovania väčší alebo rovný než zvolený dátum od (`LOG.DATETIME >= @dateFrom`) a zároveň menší alebo rovný než zvolený dátum do (`LOG.DATETIME <= @dateTo`) [FR16.3]
  - pri porovnávaní sa berie do úvahy len dátum, čas sa ignoruje

Level [E03] (rozbaľovacia ponuka)

- Zobrazenie:
  - predvolene sa zobrazuje lokalizovaný popis „*Typ udalosti*“
  - v ponuke sa zobrazujú názvy typov udalostí z {FATAL, ERROR, WARN, INFO, DEBUG}
- Ovládanie: výberom možnosti sa rozšíri filtračná podmienka o zobrazenie len logovaných udalostí, ktoré sú vybraného typu (`LOG.LEVEL` = E03) [FR16.2]

Hľadať [E05] (vyhľadávacie pole)

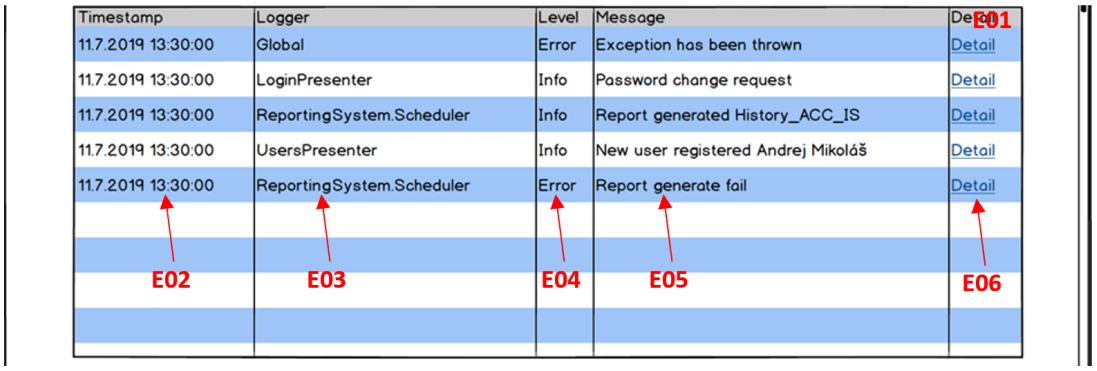
- Zobrazenie: vyhľadávacie pole s nápoveďou s lokalizovaným textom „*Hľadať*“

- Ovládanie:
  - po vpísaní textu sa na konci poľa zobrazí tlačidlo pre vymazanie textu
    - kliknutím sa vymaže zadaný text z vyhľadávacieho poľa a dôjde k zrušeniu filtračnej podmienky podľa zadaného textu a tlačidlo sa skryje
  - vpísaním a potvrdením textu sa rozšíri filtračná podmienka o zobrazenie len logovaných udalostí, u ktorých sa všetky zadané slová vyskytujú ako podreťazec v správe udalosti (`LOG.MESSAGE`) alebo v názve loggera (`LOG.LOGGER`) [FR16.4]

Zrušiť filtre [E07] (tlačidlo)

- Zobrazenie: tlačidlo s lokalizovaným textom „*Zrušiť filtre*“
- Ovládanie:
  - kliknutím dôjde k zrušeniu všetkých filtračných podmienok a zobrazeniu všetkých logovaných udalostí
  - zároveň dôjde k nastaveniu predvolených hodnôt na všetkých položkách filtračnej lišty a vymazaniu textu z vyhľadávacieho poľa

Pracovná plocha:



Obr. 25: UC16 - pracovná plocha

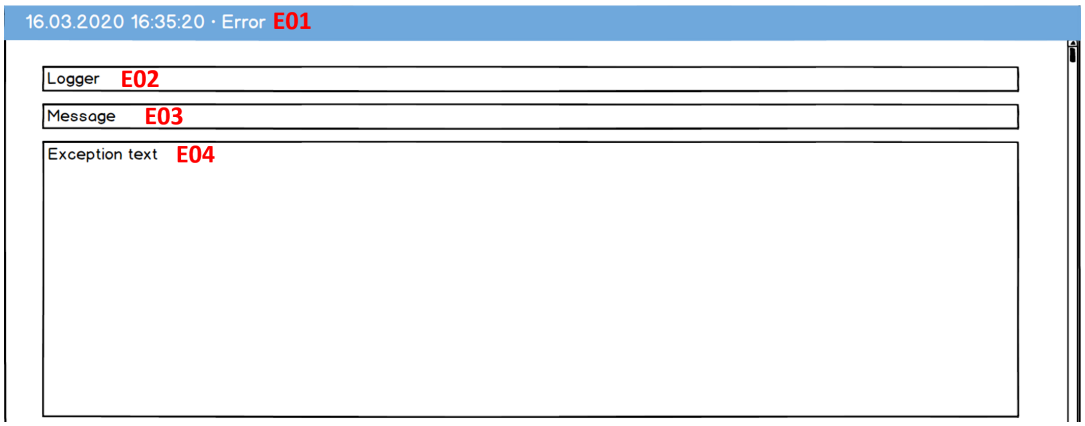
Logované udalosti [E01] (tabuľka)

- Zobrazenie:
  - tabuľka s piatimi stĺpcami, každý s vlastnou hlavičkou
  - každá hlavička obsahuje lokalizovaný názov stĺpca
  - tabuľka obsahuje všetky záznamy z tabuľky `LOG` zoradené od najnovšieho záznamu (`ORDER BY LOG.DATETIME DESC`). [FR16.1]
  - každý riadok predstavuje jeden záznam s atribútmi:
    - Dátum vzniku [E02] `LOG.DATETIME`: vo formáte „*DD.MM.YYYY HH:MM:SS*“
    - Názov loggera [E03] `LOG.LOGGER`
    - Level [E04] `LOG.LEVEL`

    - Správa [E05] `LOG.MESSAGE`
    - Detail [E06] tlačidlo s lokalizovaným textom „*Detail*“
        - kliknutím dôjde k presmerovaniu na UC17 Detail logovanej udalosti
        - ako parameter sa pošle `LOG.ID_LOG`

UC17 Detail logovanej udalosti

Aktéri: administrátor
Predpoklady: aktér je prihlásený v systéme
Spúšťač: administrátor chce zobraziť detail zaznamenanej udalosti v systéme
Použité tabuľky: `LOG`
Do UC vstupuje identifikátor logovanej udalosti (@IdLog), na základe ktorého sa načíta celý záznam `LOG` [FR17.1]
Pracovná plocha:



Obr. 26: UC17 - pracovná plocha

Informácie o udalosti [E01] (text)

- Zobrazenie: dátum a čas udalosti a level závažnosti (`LOG.DATETIME, LOG.LEVEL`)

Detail udalosti [E02, E03, E04] (textové polia bez možnosti editovania)

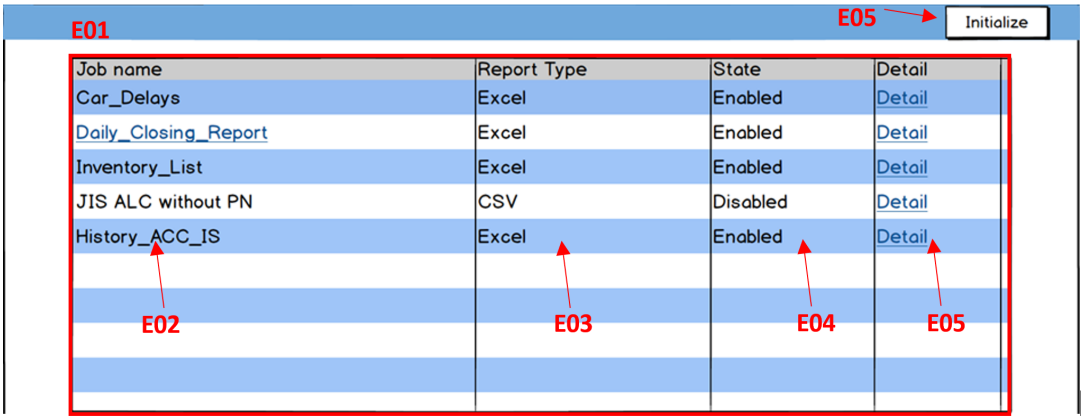
- E02: obsahuje hodnotu atribútu `LOG.LOGGER`
- E03: obsahuje hodnotu atribútu `LOG.MESSAGE`
- E04: obsahuje hodnotu atribútu `LOG.EXCEPTION`

UC7 Evidencia definícií reportov

Aktéri: administrátor
Predpoklady: aktér je prihlásený v systéme
Spúšťač: administrátor chce zobraziť nadefinované definície reportov

Použité tabuľky: `REPORT_DEFINITION, REPORT`

Pracovná plocha:



Obr. 27: UC7 - pracovná plocha

Definície reportov [E01] (tabuľka)

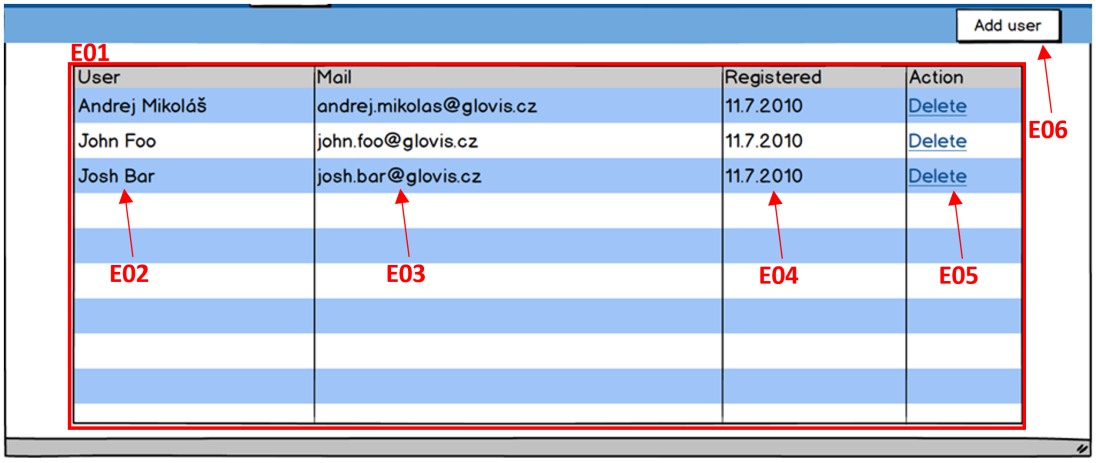
- Zobrazenie:
  - tabuľka so štyrmi stĺpcami, každý s vlastnou hlavičkou
  - každá hlavička obsahuje lokalizovaný názov stĺpca
  - tabuľka obsahuje všetky záznamy z tabuľky `REPORT_DEFINITION` zoradené podľa názvu (`ORDER BY REPORT_DEFINITION.NAME ASC`) [FR7.1]
  - každý riadok predstavuje jeden záznam s atribútmi:
    - Názov definície [E02] `REPORT_DEFINITION.NAME`
    - Typ reportu [E03] `REPORT_DEFINITION.REPORT_TYPE`
    - Stav [E04] predstavuje stav hodnoty atribútu `REPORT_DEFINITION.ACTIVE_B`
      - `ACTIVE_B = 1`: zobrazuje sa lokalizovaný text „*Aktívny*“
      - `ACTIVE_B = 0`: zobrazuje sa lokalizovaný text „*Neaktívny*“
    - Detail [E05] tlačidlo s lokalizovaným textom „*Detail*“
      - kliknutím dôjde k presmerovaniu na UC10 Správa definície reportu
      - ako parameter sa pošle `REPORT_DEFINITION.ID_REPORT_DEF`

Inicializovať [E05] (tlačidlo)

- Zobrazenie: tlačidlo s lokalizovaným textom „*Inicializovať*“
- Ovládanie:
  - kliknutím sa spustí proces inicializácie definícií reportov
  - pomocou reflexie sa prechádzajú všetky triedy v špecifickom balíčku predstavujúcom definície reportov [FR8.1]
  - pre všetky pridané triedy sa založia nové záznamy v tabuľke `REPORT_DEFINITION`, prípadne sa aktualizujú existujúce ak došlo iba ku zmene [FR8.2]

UC6 Evidencia registrovaných užívateľov

Aktéri: administrátor
Predpoklady: aktér je prihlásený v systéme
Spúšťač: administrátor chce zobraziť všetkých užívateľov, ktorí sú registrovaní v systéme
Použité tabuľky: `REGISTERED_USER`
Pracovná plocha:



Obr. 28: UC6 - pracovná plocha

Zoznam užívateľov [E01] (tabuľka)

- Zobrazenie:
  - tabuľka so štyrmi stĺpcami, každý s vlastnou hlavičkou
  - každá hlavička obsahuje lokalizovaný názov stĺpca
  - tabuľka obsahuje všetkých platných užívateľov z tabuľky `REGISTERED_USER WHERE VALID_B=1` zoradených podľa priezviska `(ORDER BY REGISTERED_USER .SURNAME ASC)` [FR6.1]
  - každý riadok predstavuje jeden záznam s atribútmi:
    * Užívateľ [E02] v tvare `REGISTERED_USER.NAME REGISTERED_USER.SURNAME`
    * Mail [E03] `REGISTERED_USER.MAIL`
    * Dátum registrácie [E04] `REGISTERED_USER.DATE_REGISTERED`: dátum vo formáte „*DD.MM.YYYY HH:MM:SS*“
    * Akcie [E05] tlačidlo s lokalizovaným textom „*Vymazať*“
      · kliknutím sa zobrazí dialóg, ktorým užívateľ potvrdí alebo zamietne zmazanie užívateľa
      · po potvrdení sa záznam užívateľa v databáze označí ako neplatný `(REGISTERED_USER.VALID_B=0)`

Pridať užívateľa [E06] (tabuľka)

- Zobrazenie: tlačidlo s lokalizovaným textom „*Pridať užívateľa*“
- Ovládanie: kliknutím dôjde k spusteniu UC5 Registrácia užívateľov

UC5 Registrácia užívateľov

Aktéri: administrátor
Predpoklady: aktér je prihlásený v systéme
Spúšťač: administrátor chce umožniť prístup do systému novému užívateľovi
Použité tabuľky: `REGISTERED_USER`
UC sa zobrazuje ako oblasť na obrazovke v UC6 Evidencia registrovaných užívateľov.
Kompozícia užívateľského rozhrania:

Obr. 29: UC5 - kompozícia užívateľského rozhrania

Meno, Priezvisko, Mail [E01, E02, E03] (editovateľné polia)

- Zobrazenie: editovateľné polia so štítkami s lokalizovaným textom „*Meno, Priezvisko, Mail*“

Zrušiť [E04] (tlačidlo)

- Zobrazenie: tlačidlo s lokalizovaným textom „*Zrušiť*“
- Ovládanie: kliknutím sa zavrie oblasť s textovými poľami bez uloženia informácii a UC končí

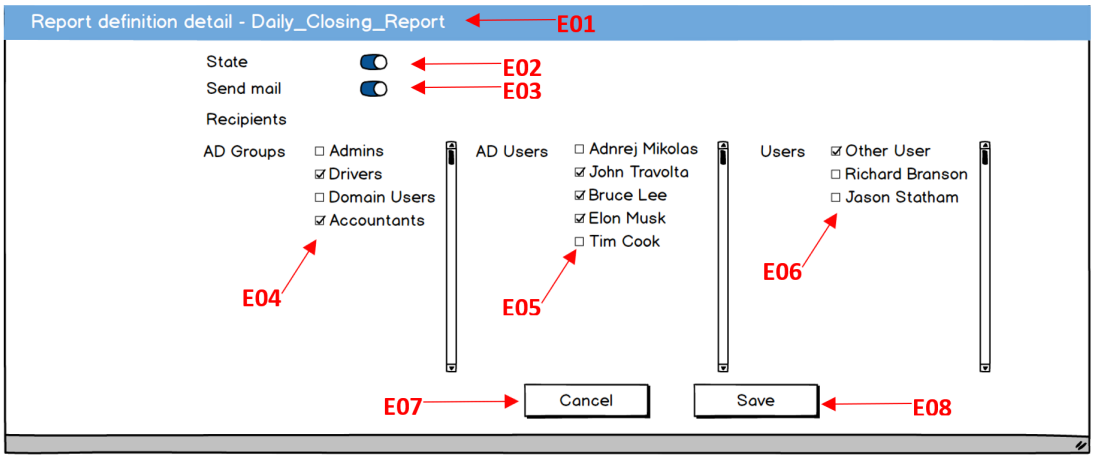
Uložiť [E05] (tlačidlo)

- Zobrazenie: tlačidlo s lokalizovaným textom „*Uložiť*“
- Ovládanie:
  - kliknutím sa overí, že všetky editovateľné polia sú vyplnené
  - v prípade, že niektoré pole je prázdne, zobrazí sa hláška o chýbajúcom údaji a UC končí
  - následne sa overí, že zadaná mailová adresa (E03) je v správnom formáte (ak je formát neplatný, zobrazí sa hláška o neplatnom maile a UC končí)
  - systém overí, že zadaný mail ešte v databáze neexistuje
    - v prípade, že mail už existuje, zobrazí sa hláška o duplicite mailovej adresy a UC končí [FR5.4]
  - systém vygeneruje nový náhodný 8-miestny reťazec čísel a znakov abecedy (@newPass) a vytvorí jeho hash hodnotu (@NewPassHash) [FR5.2]

- založí sa nový záznam registrovaného užívateľa v databáze [FR5.1]:
  - `REGISTERED_USER.NAME` = hodnota poľa E01
  - `REGISTERED_USER.SURNAME` = hodnota poľa E02
  - `REGISTERED_USER.MAIL` = hodnota poľa E03
  - `REGISTERED_USER.PASSWORD` = @NewPassHash
  - `REGISTERED_USER.INITIAL_PASSWORD_B` = 1
  - `REGISTERED_USER.DATE_REGISTERED` = aktuálny dátum
  - `REGISTERED_USER.ROLE` = 'USER'
- pomocou komponenty pre odosielanie mailov sa vytvorí a odošle nová správa na mailovú adresu nového užívateľa (`REGISTERED_USER.MAIL`) [FR5.3]
- správa bude obsahovať informáciu o registrovaní do systému a počiatočné heslo (@NewPass)
- systém zaeviduje registráciu užívateľa do logu [FR20.1]
- oblasť s textovými poľami sa zavrie, zobrazí sa hláška o úspešnom vytvorení užívateľa, aktualizuje sa zoznam registrovaných užívateľov a UC končí

UC10 Správa definície reportu

Aktéri: administrátor
Predpoklady: aktér je prihlásený v systéme
Spúšťač: administrátor počas UC7 Evidencia definícií reportov klikol na jeden záznam v tabuľke definícii reportov
Použité tabuľky: `REGISTERED_USER, REPORT_DEFINITION, REPORT_DEFINITION_RECIPIENT`
Do UC vstupuje identifikátor definície reportu (@IdReportDef), na základe ktorého sa načíta celý záznam, vrátane jej prijímateľov. Všetky zmeny vykonané v tomto okne sa do databázy zapíšu až po potvrdení užívateľom pomocou tlačidla.
Pracovná plocha:



Obr. 30: UC10 - pracovná plocha

Názov definície [E01] (text)

- Zobrazenie: názov definície reportu (`REPORT_DEFINITION.NAME`) s prefixom s lokalizovaným textom „*Detail definície reportu –*“

Stav [E02] (checkbox)

- Zobrazenie: checkbox s popisom s lokalizovaným textom „*Stav*“
- Ovládanie: zmenou stavu sa zapína a vypína generovanie reportov z danej definície (`REPORT_DEFINITION.ACTIVE_B`) [FR10.1]

Odosielať maily [E03] (checkbox)

- Zobrazenie: checkbox s popisom s lokalizovaným textom „*Odosielať maily*“
- Ovládanie: zmenou stavu sa zapína a vypína odosielanie informatívnych mailov o vygenerovaní reportu cieľovej skupine užívateľov (`REPORT_DEFINITION.SEND_MAIL_B`) [FR10.2]

Zoznam AD skupín [E04] (zoznam so zaškrtávacími políčkami)

- Zobrazenie:
  - zoznam obsahuje všetky skupiny vo firemnej doméne získané z AD [FR9.2]
  - položky zoznamu, ktoré predstavujú skupinu, ktorá má k definícii väzbu, má začiarknuté políčko (`REPORT_DEFINITION_RECIPIENT.AD_GROUP_ID`, väzba na definíciu cez `REPORT_DEFINITION.ID_REPORT_DEF=REPORT_DEFINITION_RECIPIENT.ID_REPORT_DEF`)
- Ovládanie: zmenou stavu políčka sa daná skupina priraďuje alebo odstraňuje od definície reportu [FR9.4]

Zoznam AD užívateľov [E05] (zoznam so zaškrtávacími políčkami)

- Zobrazenie:
  - zoznam obsahuje všetkých užívateľov vo firemnej doméne získané z AD [FR9.1]
  - položky zoznamu, ktoré predstavujú užívateľa, ktorý má k definícii väzbu, má začiarknuté políčko (`REPORT_DEFINITION_RECIPIENT.AD_USER_ID`, väzba na definíciu cez `REPORT_DEFINITION.ID_REPORT_DEF=REPORT_DEFINITION_RECIPIENT.ID_REPORT_DEF`)
- Ovládanie: zmenou stavu políčka sa daný užívateľ priraďuje alebo odstraňuje od definície reportu [FR9.4]

Zoznam registrovaných užívateľov [E04] (zoznam so zaškrtávacími políčkami)

- Zobrazenie:
  - zoznam obsahuje všetkých registrovaných užívateľov (`REGISTERED_USER`) [FR9.3]

  - položky zoznamu, ktoré predstavujú užívateľa, ktorý má k definícii väzbu, má začiarknuté políčko (`REPORT_DEFINITION_RECIPIENT.ID_USER`, väzba na definíciu cez `REPORT_DEFINITION.ID_REPORT_DEF=REPORT_DEFINITION_RECIPIENT.ID_REPORT_DEF`)
- Ovládanie: zmenou stavu políčka sa daný užívateľ priraďuje alebo odstraňuje od definície reportu [FR9.4]

Zrušiť [E08] (tlačidlo)

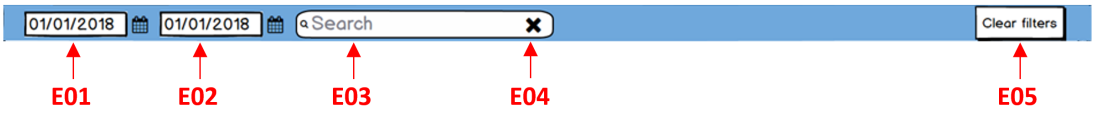
- Zobrazenie: tlačidlo s lokalizovaným textom „*Zrušiť*“
- Ovládanie: kliknutím dôjde k presmerovaniu späť na UC7 Evidencia definícií reportov bez uloženia zmien a UC končí

Uložiť [E09] (tlačidlo)

- Zobrazenie: tlačidlo s lokalizovaným textom „*Uložiť*“
- Ovládanie: kliknutím dôjde k uloženiu všetkých vykonaných zmien v definícii reportu a následnému presmerovaniu späť na UC7 Evidencia definícií reportov

UC13 Evidencia dostupných reportov

Aktéri: bežný užívateľ, registrovaný užívateľ
Predpoklady: aktér je prihlásený v systéme
Spúšťač: užívateľ si chce zobraziť vygenerované reporty ku ktorým má prístup
Použité tabuľky: `REGISTERED_USER`, `REPORT_DEFINITION`, `REPORT_DEFINITION_RECIPIENT`, `REPORT`
Pri spustení okna sa z užívateľskej relácie načíta id prihláseného užívateľa @IdUser (identifikátor v tabuľke `REGISTERED_USER` alebo identifikátor užívateľa v AD) a v prípade AD užívateľa aj zoznam id skupín v doméne AD, do ktorej je zaradený (@GroupId).
Lišta filtrov:



Obr. 31: UC13 - lišta filtrov

Dátum od, dátum do [E01, E02] (komponent pre výber dátumu)

- Zobrazenie: predvolene je zobrazený aktuálny dátum
- Ovládanie:
  - kliknutím sa otvorí komponent pre výber dátumu z kalendára

- potvrdením nového dátumu (`@dateFrom` alebo `@dateTo`) sa rozšíri filtračná podmienka o zobrazenie len reportov, ktoré majú dátum vygenerovania väčší alebo rovný než zvolený dátum od (`LOG.DATETIME >= @dateFrom`) a zároveň menší alebo rovný než zvolený dátum do (`LOG.DATETIME <= @dateTo`) [FR13.3]
- pri porovnávaní sa berie do úvahy len dátum, čas sa ignoruje

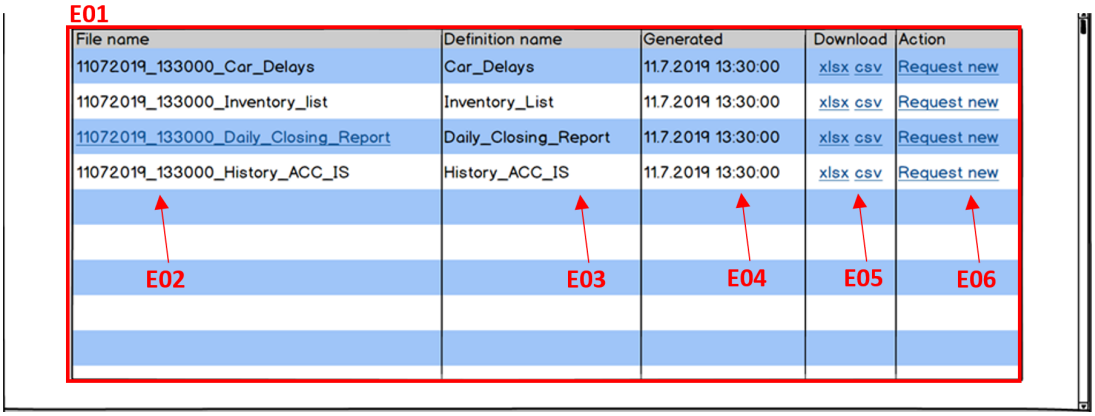
Hľadať [E03] (vyhľadávacie pole)

- Zobrazenie: vyhľadávacie pole s nápoveďou s lokalizovaným textom „*Hľadať*“
- Ovládanie:
  - po vpísaní textu sa zobrazí tlačidlo pre vymazanie textu E04
    - kliknutím sa vymaže zadaný text z vyhľadávacieho poľa a dôjde k zrušeniu filtračnej podmienky podľa zadaného textu a tlačidlo sa skryje
  - vpísaním a potvrdením textu sa rozšíri filtračná podmienka o zobrazenie len reportov, u ktorých sa všetky zadané slová vyskytujú ako podreťazec v názve súboru alebo názve definície, z ktorej vznikol (`REPORT.FILE_NAME` alebo `REPORT_DEFINITION.NAME`, väzba na report cez `REPORT.ID_REPORT_DEF=REPORT_DEFINITION.ID_REPORT_DEF`) [FR13.2]

Zrušiť filtre [E05] (tlačidlo)

- Zobrazenie: tlačidlo s lokalizovaným textom „*Zrušiť filtre*“
- Ovládanie:
  - kliknutím dôjde k zrušeniu všetkých filtračných podmienok a zobrazeniu všetkých reportov
  - zároveň dôjde k nastaveniu predvolených hodnôt na všetkých položkách filtračnej lišty a vymazaniu textu z vyhľadávacieho poľa

Pracovná plocha:



Obr. 32: UC13 - pracovná plocha

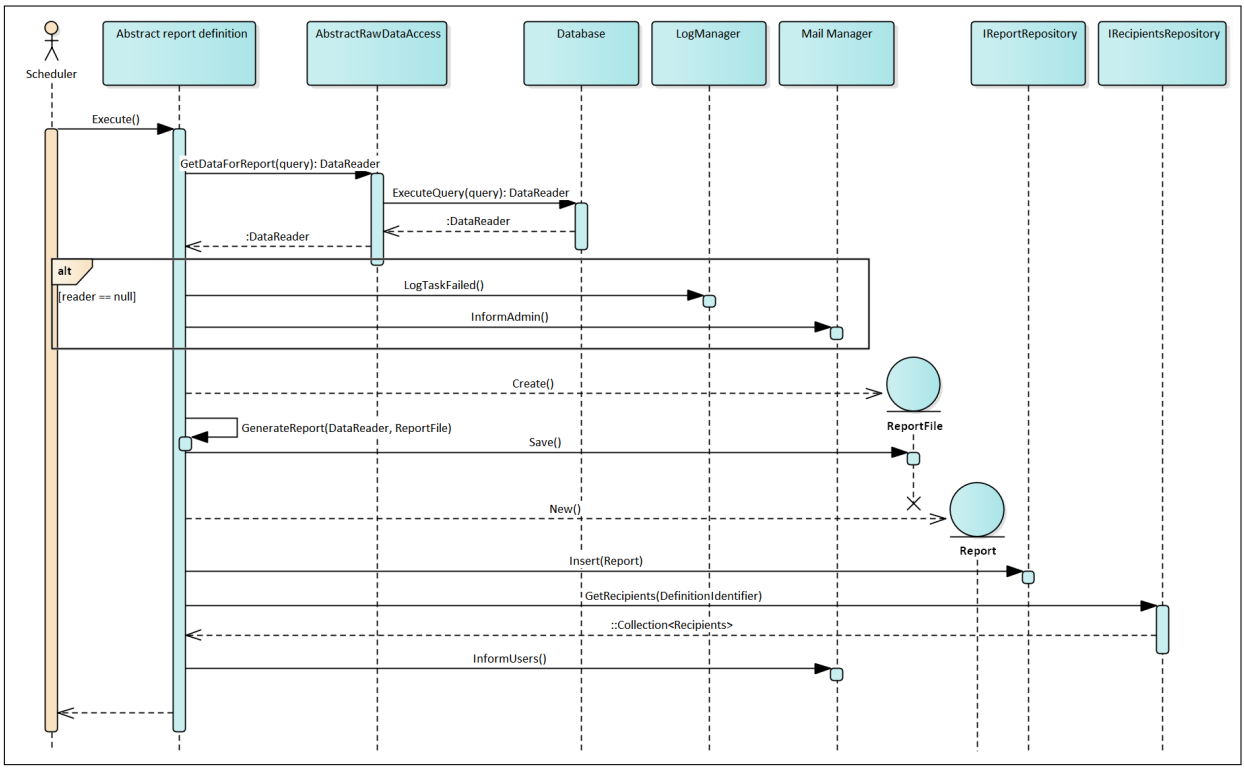
Reporty [E01] (tabuľka)

- Zobrazenie:
  - tabuľka s piatimi stĺpcami, každý s vlastnou hlavičkou.
  - každá hlavička obsahuje lokalizovaný názov stĺpca
  - tabuľka obsahuje všetky reporty, ktoré boli vygenerované z definície, ku ktorej má užívateľ prístup [FR13.1]
    * väzba reportu na jeho definíciu cez `REPORT_DEFINITION.ID_REPORT_DEF=REPORT.ID_REPORT_DEF`, väzba definície na prijímateľa cez `REPORT_DEFINITION.ID_REPORT_DEF=REPORT_DEFINITION_RECIPIENT.ID_REPORT_DEF`
    * v prípade registrovaného užívateľa sa vyberajú len záznamy podľa `REPORT_DEFINITION_RECIPIENT.ID_USER=@IdUser`
    * v prípade AD užívateľa sa vyberajú záznamy podľa `REPORT_DEFINITION_RECIPIENT.AD_USER_ID=@IdUser` a následne `REPORT_DEFINITION_RECIPIENT.AD_GROUP_ID=@GroupId` (duplicitné záznamy sa vylučujú)
  - každý riadok predstavuje jeden záznam s atribútmi:
    * Názov súboru [E02]: `REPORT.FILE_NAME`
    * Názov definície reportu [E03]: `REPORT_DEFINITION.NAME`, väzba na report cez `REPORT_DEFINITION.ID_REPORT_DEF = REPORT.ID_REPORT_DEF`
    * Dátum vygenerovania [E04]: `REPORT.CREATE_DATE` vo formáte „*DD.MM.YYYY HH:MM:SS*“
    * Stiahnuť [E05]:
      · tlačidlo s lokalizovaným textom „*Stiahnuť*“
      · kliknutím sa spustí komponent pre stiahnutie súboru z úložiska z adresy `REPORT.FILE_PATH` [FR14.1]
    * Vyžiadať nový [E06]:
      · tlačidlo s lokalizovaným textom „*Vyžiadať nový*“
      · kliknutím sa spustí okamžité vygenerovanie reportu z definície, z ktorej vznikol daný report (`REPORT_DEFINITION`, väzba na zobrazený report cez `REPORT.ID_REPORT_DEF = REPORT_DEFINITION.ID_REPORT_DEF`) [FR15.1]
      · podľa nastavení odosielania mailov sa prípadne odošle mail cieľovej skupine užívateľov a taktiež sa vygenerovaný report sprístupní cieľovej skupine

3.3.5 Procesný pohľad

Sekvenčný diagram zobrazuje procesy vykonávané sekvenčne, ktoré sú usporiadané v čase a vytvára sa použitím jazyka UML. Z procesov v tomto systéme som si zvolil pre ukážku proces generovania reportu. Spúšťačom procesu je plánovač, ktorý nad definíciou reportu spúšťa metódu `void Execute()`. Nasleduje získanie dát z databázy na základe SQL dotazu. V prípade chyby pri získavaní dát je toto zlyhanie zaznamenané do logu a administrátor je informovaný mailom. Následne sa zo získaných dát vytvorí výstupný súbor, informácie o reporte sa uložia do databázy

a cieľovým užívateľom je rozposlaná mailová správa. Tento proces je zachytený na sekvenčnom diagrame na obrázku 33.



Obr. 33: Sekvenčný diagram procesu generovania reportu

4 Implementácia

V tejto časti práce sa zameriam na popis implementácie vybraných funkcionalít výslednej aplikácie. Zameriam sa najmä na prihlasovanie do systému, ktoré bolo implementačne najviac náročné, ale tiež napríklad na implementáciu použitia vzoru Model View Presenter, prístupu ku databáze a generovanie reportov.

4.1 Zabezpečenie aplikácie

Pre prihlasovanie do aplikácie boli vyžadované 3 spôsoby:

- automaticky použitím Windows účtov zamestnancov,
- pomocou mailu a hesla,
- manuálnym zadaním prihlasovacích údajov z Windows účtu.

V praxi to znamenalo skombinovať *Windows* a *Forms Authentication*, čo však samotné ASP.NET natívnym spôsobom nepodporuje. Preto som musel pristúpiť k vlastnej implementácii prihlasovania. V konfiguračnom súbore aplikácie som ponechal `authentication mode="Forms"`, teda prihlasovanie cez formulár.

`IIdentity` a `IPrincipal`

Pre možnosť kombinácie spomínaných dvoch typov prihlasovania bolo potrebné implementovať rozhrania `IIdentity` a `IPrincipal` z menného priestoru `System.Security.Principal` a teda vytvoriť vlastné objekty pre potreby prihlasovania. Objekt typu `Identity` obaľuje informácie o užívateľovi, ktorý sa validuje. Jeho rozhranie definuje prístupové metódy pre meno užívateľa, typ autentifikácie a príznak, či je autentifikovaný. Objekt typu `Principal` reprezentuje bezpečnostný kontext, pod ktorým je kód spustený a jeho rozhranie definuje prístupové metódy pre pridružený objekt `Identity` a tiež metódu `bool IsInRole(string role)`. Táto metóda rozhoduje, či užívateľ má danú rolu [21]. Tento prístup sa využíva pri autorizácii na základe role, ktorú som využil v tomto systéme.

Do vlastnej implementácie `IIdentity` som doplnil vlastnosti pre registrovaného a pre doménového užívateľa. V jednej inštancii sa vždy nachádza len jeden zo spomínaných dvoch typov užívateľov. Pre zjednodušenie overovania role som doplnil ešte jednu vlastnosť pre rolu užívateľa. Preťažil som konštruktor, a ako ich parameter sa predá buď registrovaný alebo doménový užívateľ, ktorého si vytvorím po úspešnej autentifikácii. K užívateľovi si zároveň získam rolu a uložím. Registrovaný užívateľ má rolu uloženú priamo v databáze. U doménového si najskôr získam všetky doménové skupiny, a ak patrí do „GLOVIS\GCZ_IT“, tak mu nastavím rolu ADMIN, inak USER. Na výpise 2 je zobrazená ukážka popisovanej triedy kde je pre názornosť ponechaný len konštruktor pre registrovaného užívateľa.

```
public class CustomUserIdentity : IIdentity {
```

```
    public IIdentity Identity { get; set; }
    public RegisteredUser RegisteredUser { get; set; }
    public ADUser AdUser { get; set; }
    public string Role { get; set; }
    public CustomUserIdentity(RegisteredUser registeredUser) {
        string username = registeredUser.Mail;
        Identity = new GenericIdentity(username);
        RegisteredUser = registeredUser;
        Role = registeredUser.Role;
    }
    ...
}
```

Výpis 2: Ukážka vlastnej implementácie rozhrania `IIdentity`

Vo vlastnej implementácii `Principal` som ako vlastnosť typu `IIdentity` použil mnou implementovanú variantu, ktorá obsahovala všetko potrebné a dokázala udržať registrovaného alebo doménového používateľa. Implementácia metódy pre overenie role potom len jednoducho porovnala rolu z parametru s rolou v tejto vlastnosti. Ukážka tejto triedy je na výpise 3.

```
public class CustomUserPrincipal : IPrincipal {
    private CustomUserIdentity _userIdentity;
    public CustomUserPrincipal(CustomUserIdentity userIdentity) {
        _userIdentity = userIdentity;
    }
    public IIdentity Identity { get { return _userIdentity; } }
    public bool IsInRole(string role) {
        return role.ToUpper().Equals(_userIdentity.Role);
    }
}
```

Výpis 3: Ukážka vlastnej implementácie rozhrania `IPrincipal`

Zabezpečenie stránok

Autorizácia na základe rolí umožňuje kontrolovať zobrazenie položiek navigačného menu podľa role prihláseného užívateľa. Túto možnosť som využil, aby som bežnému užívateľovi skryl z menu stránky určené len pre administrátora. Nastavenie spočívalo vo vytvorení súboru `Web.sitemap`, kde sa zaregistrovali všetky stránky a v konfiguračnom súbore aplikácie sa táto možnosť povolila. Administrátorove a užívateľove stránky som presunul do osobitných adresárov. V každom adresári existuje súbor `Web.config` kde je uvedené, ktoré role sú pre stránky v danom adresári povolené a ktoré nie. Jeho ukážka je na výpise 4. Ukážka súboru `Web.sitemap` je na výpise 5.

```
<configuration>
  <system.web>
    <authorization>
      <allow roles="ADMIN"/>
      <deny users="*"/>
    </authorization>
  </system.web>
</configuration>
```

Výpis 4: Ukážka súboru Web.config z adresára pre administrátorove stránky

```
<siteMap xmlns="http://schemas.microsoft.com/AspNet/SiteMap-File-1.0" >
  <siteMapNode url="Default.aspx" title="Login" description="">
    <siteMapNode url="User/Reports.aspx" title="Reports" description="" />
    <siteMapNode url="Admin/ReportDefinitions.aspx" title="Report Definitions"
       description="" />
    <siteMapNode url="Admin/Logs.aspx" title="Logs" description="" />
    <siteMapNode url="Admin/Users.aspx" title="Registered Users" description=""
        />
  </siteMapNode>
</siteMap>
```

Výpis 5: Ukážka súboru Web.sitemap

4.2 Model View Presenter

Implementácia tohto architektonického vzoru je veľmi intuitívna a jednoduchá. Vrstvu *Model* tvorí dátová vrstva a triedy repozitárov, ktorého ukážka je na výpise 6. Ukážka rozhrania, cez ktoré komunikuje *Presenter* a konkrétna stránka vo *View* je zobrazená na výpise 7. Toto rozhranie implementuje prezentačná logika stránky v prezentačnej vrstve a jeho ukážka je na výpise 8. Tá potrebuje len referenciu na *Presenter*, ktorý si sám vytvorí a ako parameter v jeho konštruktore poskytuje referenciu na seba. Potom je možné volať metódy *Presentera*. Samotný *Presenter* obsahuje referenciu na *View*, ktorý mu príde v konštruktore a tiež referenciu na repozitáre, predstavujúce vrstvu *Model*. Ukážka triedy *Presentera* je na výpise 9. V tejto ukážke je metóda `void Initialize(long reportDefId, bool isPostBack)`, ktorá je volaná z *View* a má na starosti získanie potrebných dát z repozitára a ich dosadenie do *View* cez rozhranie. Na diagrame na obrázku 34 je ukážka rozmiestnenia jednotlivých tried vzoru MVP v rámci logického modelu systému.

```
public class ReportDefinitionRepository : IReportDefinitionRepository {
```

```
    ...
    public ReportDefinition Get(long id) {
        using (MySqlContext context = new MySqlContext()) {
            return context.ReportDefinitions.Find(id);
        }
    }
    ...
}
```

Výpis 6: Ukážka súboru ReportDefinitionRepository.cs predstavujúceho Model v architektúre MVP

```
public interface IEditReportDefinitionView {
    string ReportDefinitionTitle { set; }
    bool SendMailState { get; set; }
    ...
}
```

Výpis 7: Ukážka súboru IEditReportDefinitionView.cs predstavujúceho rozhranie cez ktoré komunikujú View a Presenter v architektúre MVP

```
public partial class EditReportDefinitionPage : System.Web.UI.Page,
    IEditReportDefintionView {
    private readonly EditReportDefinitionPresenter _presenter;
    public EditReportDefinitionPage() {
        this._presenter = new EditReportDefinitionPresenter(this);
    }
    protected void Page_Load(object sender, EventArgs e) {
        try {
            long definitionId = long.Parse(Request.QueryString["definitionId"]);
            _presenter.Initialize(definitionId, IsPostBack);
        }
        catch (Exception) { ... }
    }
    public string ReportDefinitionTitle {
        set { Label_definitionName.Text = value; }
    }
    public bool SendMailState {
        get { return CheckBox_mail.Checked; }
        set { CheckBox_mail.Checked = value; }
    }
```

```
}
```

Výpis 8: Ukážka súboru EditReportDefinitionPage.aspx.cs predstavujúceho View v architektúre MVP

```
public class EditReportDefinitionPresenter {
    ...
    private readonly IEditReportDefintionView _view;
    private readonly IReportDefinitionRepository _reportDefRepository;
    private ReportDefinition _definition;
    public EditReportDefinitionPresenter(IEditReportDefintionView _view) {
        this._view = _view;
        _reportDefRepository = RepositoryFactory.GetReportDefinitionRepository
            ();
        ...
    }
    public void Initialize(long reportDefId, bool isPostBack) {
        _definition = _reportDefRepository.Get(reportDefId);
        if (_definition == null) { return; }
        if (isPostBack) { return; }
        _view.ReportDefinitionTitle = string.Format("{0} - {1}", Language.
            report_def_title, _definition.Name);
        _view.SendMailState = _definition.SendMail;
        ...
    }
}
```

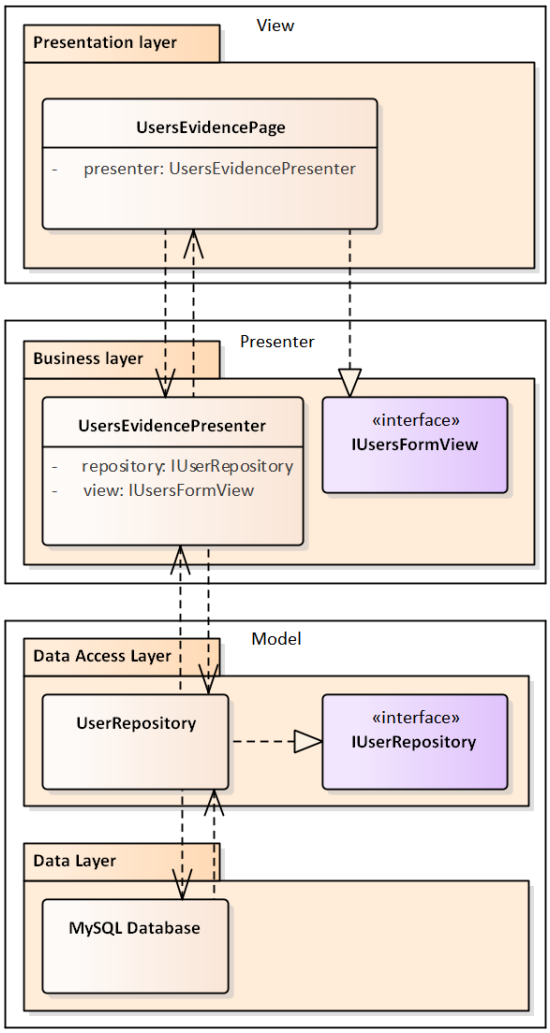
Výpis 9: Ukážka súboru EditReportDefinitionPresenter.cs predstavujúceho Presenter v architektúre MVP

4.3 Prístup ku dátam

Entity Framework 6 je objektovo-relačný mapovací nástroj, ktorý efektívne zmenšuje priepasť medzi relačnými a objektovo orientovanými svetmi[9]. Vývojárom umožňuje jednoducho pracovať s dátami v relačnej databáze pomocou silne typovaných .NET objektov, bez potreby písania zdĺhavého kódu. [22]

Pre vytvorenie databázovej štruktúry som využil prístupu *code first*, čo znamená, že z existujúcich doménových objektov, ktoré majú nastavené vhodné anotácie, sa vygenerujú skripty

[9] `https://www.nuget.org/packages/EntityFramework/`



Obr. 34: Ukážka komunikácie vrstiev vzoru MVP

pre vytvorenie databázového modelu. Vytvorenie týchto skriptov sa nazýva migrácia. V prípade úpravy doménového modelu je možné tieto zmeny premietnuť aj do databázy ďalšou migráciou.

Pre prístup ku dátam v MySQL databáze je využitý vzor *Repository*. Každý repozitár zastrešuje databázové operácie nad jednou tabuľkou. `IRepository` je základným rozhraním pre všetky repozitáre a predpisuje základné CRUD operácie nad dátami. Aplikácia bola implementovaná s ohľadom na možnú budúcu zmenu dátového úložiska, ktoré by vyžadovalo inú implementáciu prístupu ku dátam. Preto je medzi základným rozhraním a konkrétnym repozitárom ďalšie rozhranie pre špecifický typ objektov s ktorými pracuje. Napríklad pre tabuľku registrovaných užívateľov exituje rozhranie `IRegisteredUsersRepository`, predpisujúce doplnkové metódy, ktoré musí konkrétny repozitár následne implementovať. Ukážka repozitáru je na výpise 6. Keďže pre každú tabuľku môže existovať viacero repozitárov (každý pre iný typ úložiska), využil som návrhový vzor *Abstract Factory* (abstraktná továreň). Ukážka práce s továrňou je v konštruktore na výpise 9.

V prípade Oracle databázy by bolo zdĺhavé a neefektívne vytvárať si dátové objekty a pracovať s ňou podobne ako s MySQL, pretože obsahuje enormné množstvo tabuliek. Pre potreby generovania reportov som potreboval len jednoduchú metódu, ktorá na základe poskytnutého SQL dotazu získa dáta v podobe dátových sád. Implementoval som si teda metódu `DbDataReader ExecuteSQLQuery(string query)` a je obsiahnutá v pomocnej triede `OracleRawDataSource`. Predpísaná je však v základnom rozhraní `IRawDataSource`, aby som mohol opäť aplikovať vzor *Abstract Factory* pre prípad budúcej zmeny úložiska. Na výpise 12 je ukážka práce s týmto typom úložiska v metóde `GetReaderFromSQLQuery()`. V prípade potreby je možné do budúcna doplniť metódu napríklad pre spúšťanie uložených databázových procedúr.

Posledným typom úložiska je Active Directory. Pre ten som si vytvoril len pomocnú triedu, ktorá mi poskytuje všetky potrebné operácie nad touto službou. Na výpise 10 je ukážka metódy, ktorá získava informácie o všetkých doménových skupinách. Ďalej táto pomocná trieda obsahuje napríklad metódu pre získanie všetkých užívateľov v doméne alebo metódu pre získanie všetkých skupín, do ktorých patrí užívateľ.

```
public static IEnumerable<ADGroup> GetAllAdGroups() {
    var adGroups = new List<ADGroup>();
    using (PrincipalSearcher ps = new PrincipalSearcher(new GroupPrincipal(new
        PrincipalContext(ContextType.Domain, "glovis.cz")))) {
        foreach (GroupPrincipal pr in ps.FindAll().OrderBy(p => p.
            SamAccountName)) {
            adGroups.Add(new ADGroup(pr));
        }
    }
    return adGroups;
}
```

Výpis 10: Ukážka metódy pre získanie všetkých užívateľov v doméne

4.4 Generovanie reportov

Pre vytváranie tabuľkových reportov som využil nástroj *ClosedXML*[10]. Jedná sa o .NET knižnicu pre čítanie, zapisovanie alebo manipuláciu s .xlsx a .xlsm súbormi pre Microsoft Excel od verzie 2007 a novšej. Jeho snahou je poskytnúť jednoduché rozhranie pre prácu so základným OpenXML API. Poskytuje širokú škálu možností pre štýlovanie dokumentu, ako napríklad podfarbovanie a zlučovanie buniek, zmena vlastností písma, vzorce, hárky a iné. [23]

Generovanie reportov som sa snažil implementovať tak, aby bolo možné do budúcna aplikáciu čo najjednoduchšie rozšíriť o ďalšie typy reportov. Rozhranie `IReportable` predpisuje základné vlastnosti, charakterizujúce definíciu reportu a tiež metódy pre naplánovanie vykonávania a

[10] `https://www.nuget.org/packages/ClosedXML/`

odosielanie mailu užívateľom. Jeho nadradené rozhranie `FluentScheduler.IJob` je popisované v sekcii 4.5. Ukážka tohto rozhrania je na výpise 11.

```
public interface IReportable : IJob {
    string DefinitionName { get; }
    string ClassIdentifier { get; }
    string FileNamePrefix { get; }
    ReportType ReportType { get; }
    void SendMailToRecipients();
    void ScheduleExecution();
}
```

Výpis 11: Ukážka rozhrania `IReportable`

Pod týmto rozhraním je abstraktná trieda obsahujúca implementáciu pomocných metód. Nasleduje vrstva abstraktných tried definujúcich akým spôsobom sa získajú dáta pre generovanie reportov. V súčasnej implementácii existuje len trieda `AbstractQueryDefinition` pre typy reportov, vznikajúcich na základe SQL dotazu. Dopĺňa abstraktné vlastnosti pre názov pripojenia ku databáze a samotný SQL dotaz a tiež implementuje metódu pre získanie dát z databázy. Do budúcna je možné doplniť abstraktnú triedu, ktorá bude získavať dáta napríklad volaním uloženej databázovej procedúry.

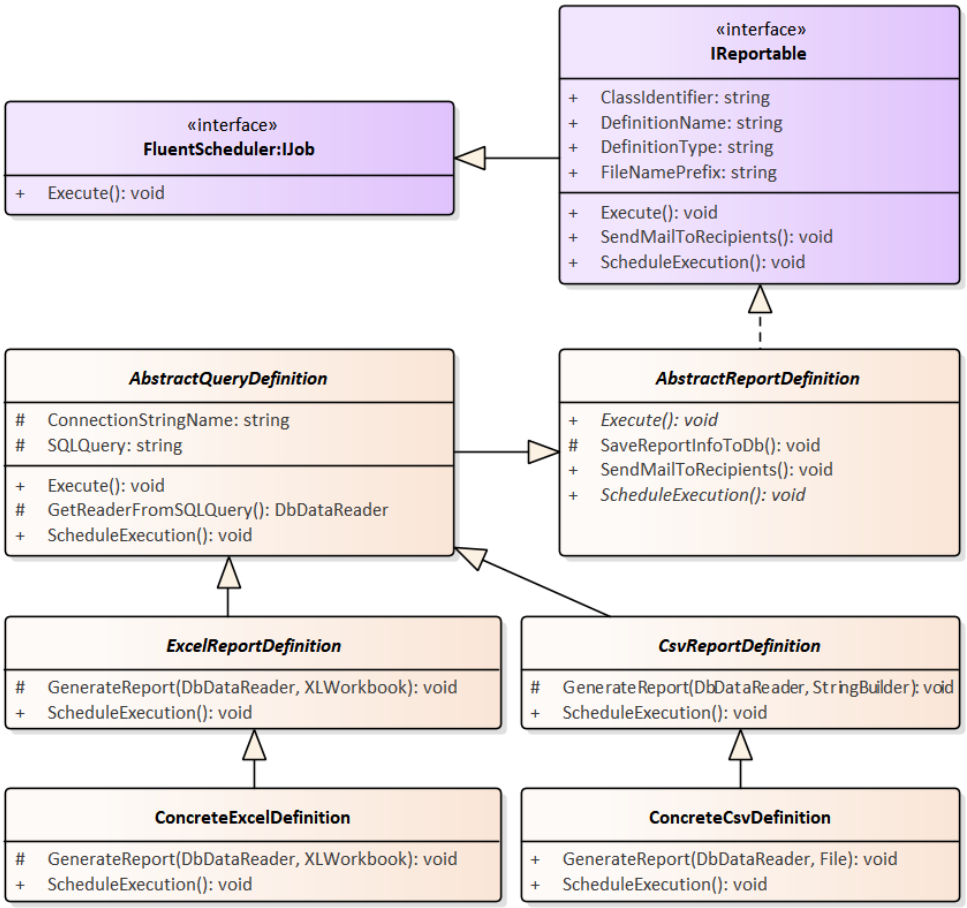
Ďalšia vrstva abstraktných tried definuje typ výstupného súboru. Implementoval som abstraktné triedy pre .xlsx a .csv súbory. Triedy v tejto vrstve implementujú virtuálnu metódu pre samotný proces generovania reportu. Tá je v základnej implementácii navrhnutá tak, že každý stĺpec zo sady získaných dát vloží do osobitného stĺpca vo výstupnom súbore a ako hlavičku nastaví názov stĺpca z databázy. Ukážka tejto metódy je na výpise 12. V prípade potreby je možné metódu pre generovanie reportu v konkrétnej definícii reportu preťažiť, a docieliť tak pokročilejšieho vzhľadu výstupného reportu.

Poslednú vrstvu tvoria konkrétne definície reportov. Pri vytváraní novej definície je potrebné len zdediť abstraktnú triedu pre požadovaný typ výstupného súboru. Vývojové prostredie si vynúti implementáciu potrebných abstraktných vlastností, čím nastavíme parametre novej definície. Taktiež sa vynúti implementácia metódy pre nastavenie frekvencie generovania. Ukážka konkrétnej definície reportu je na výpise 13. Na obrázku 35 je zobrazená popisovaná hierarchia tried a rozhraní pre definície reportov.

```
public abstract class ExcelReportDefinition : AbstractQueryDefinition {
    private static readonly ILog log = LogManager.GetLogger(MethodBase.
        GetCurrentMethod().DeclaringType);
    public override ReportType ReportType { get { return ReportType.EXCEL; } }
    public void Execute() {
        ...
        DbDataReader reader = GetReaderFromSQLQuery();
```

```
        XLWorkbook workbook = new XLWorkbook();
        try {...
            GenerateReport(reader, workbook);
            ... }
        catch (Exception e) {
            log.Error($"Failed to generate report: {FileName} - an exception has
                occured.", e);
        }
        ...
    }
    virtual void GenerateReport(DbDataReader reader, XLWorkbook workbook) {
        var worksheet = workbook.Worksheets.Add(FileNamePrefix);
        AddHeadersToExcel(reader, worksheet);
        int columnsCount = reader.FieldCount;
        int rowIndex = 2;
        while (reader.Read()) {
            for (int i = 0; i < columnsCount; i++) {
                worksheet.Cell(rowIndex, i + 1).Value = reader[i].ToString();
            }
            rowIndex++;
        }
    }
    protected DbDataReader GetReaderFromSQLQuery() {
        _dataAccess = RawDataSourceFactory.GetRawDataSource(
            ConnectionStringName);
        var reader = _dataAccess.ExecuteSQLQuery(SQLQuery);
        if (reader == null) {
            ..
            return null;
        }
        return reader;
    }
}
```

Výpis 12: Ukážka triedy `ExcelReportDefinition`



Obr. 35: Triedny diagram hierarchie tried pre definície reportov

```
public class TestReportDefinition : ExcelReportDefinition {
    public override string ClassIdentifier { get { return "60E0E065-9B90-4836-
        AD7F-5862EF519FBE"; } }
    public override string FileNamePrefix { get { return "TEST_REPORT"; } }
    public override string DefinitionName { get { return "Test report
        defininition"; } }
    public override string SQLQuery { get { return "SELECT * FROM ELIS.
        TRAIN_WAGON"; } }
    protected override string ConnectionStringName { get { return "ELIS"; } }
    ...
    public override void ScheduleExecution() {
        ScheduledJobRegistry.Instance.Schedule<TestReportDefinition>().
            ToRunEvery(5).Minutes();
        ScheduledJobRegistry.Instance.Schedule<TestReportDefinition>().
            ToRunEvery(1).Days().At(21, 15);
```

```
        ScheduledJobRegistry.Instance.Schedule<TestReportDefinition>().
            ToRunOnceIn(5).Seconds();
        ScheduledJobRegistry.Instance.Schedule<TestReportDefinition>().ToRunNow
            ().AndEvery(1).Months().OnTheFirst(DayOfWeek.Monday).At(3, 0);
    }
}
```

Výpis 13: Ukážka triedy konkrétnej definície reportu

4.5 Plánovanie generovania

Pre účely pravidelného generovania som použil knižnicu *FluentScheduler*[11]. Podporuje spúšťanie úloh v stanovenom čase alebo v pravidelných intervaloch a tiež paralelný beh rovnakej úlohy. Hlavnou výhodou tejto knižnice je skutočne jednoduché použitie a široké možnosti nastavení.

Základom konfigurácie úloh je trieda `Registry`. Jednotlivé plánované úlohy predstavujú inštancie triedy `Action` alebo triedy implementujúce rozhranie `IJob`, ktoré predpisuje metódu `void Execute()`. Táto metóda je vykonávaná plánovačom. Úlohy sa registrujú buď priamo v triede `Registry`, alebo v jej potomkovi, ktorého si vytvoríme. Nakoniec je potrebné ešte plánovač inicializovať a spustiť ho jeho statickou metódou `void JobManager.Initialize(params Registry[] registries)`. Samotný plánovač ďalej poskytuje metódy napríklad pre zastavenie, spustenie všetkých úloh alebo výpis aktuálne bežiacich. [24]

V prípade tohto systému som pre vytvorenie úlohy používal rozhranie `IJob`, ktoré implementuje abstraktná nadtrieda pre všetky definície. V rozhraní `IReportable` som si predpísal metódu `void ScheduleExecution()`, v ktorej sa do konfiguračnej triedy zaregistruje daná definícia ako nová úloha. Spúšťanie každej definície je teda možné nastaviť na viacero časov. Táto metóda je zobrazená na výpise 13 a zobrazuje rôzne možnosti nastavenia času spustenia. Pre konfiguračnú triedu `ScheduledJobRegistry` som využil návrhový vzor *Singleton*, čo mi dovoľuje registrovať nové úlohy vždy do jednej existujúcej inštancie.

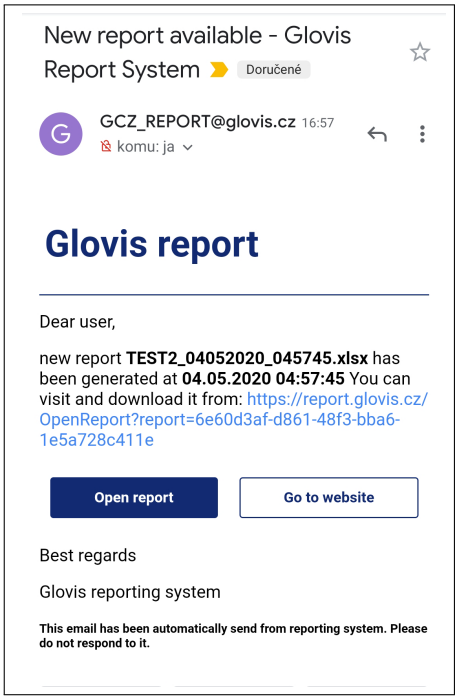
4.6 Distribúcia reportov užívateľom

Spôsob sprístupňovania reportov vzniknutých z konkrétnych definícii bol popisovaný už v doménovom pohľade pri návrhu aplikácie v sekcii 3.3.3. Po vygenerovaní reportu sa v závislosti na nastavení danej definície ešte rozpošle mailová správa cieľovým užívateľom. Správa obsahuje odkaz do reportovacieho systému na konkrétny report, ale tiež len na hlavnú stránku systému.

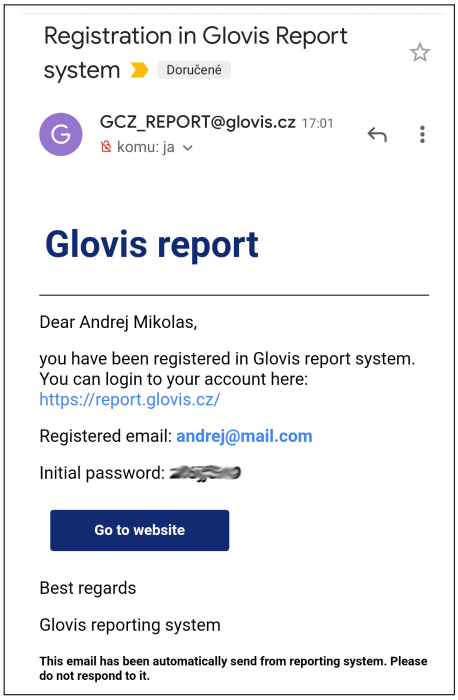
Pre tieto potreby som si pripravil HTML šablónu pre obsah mailovej správy, v ktorej sú premenné údaje nahradené zástupnými reťazcami. Pre odoslanie novej správy sa šablóna načíta a zástupné reťazce sa zamenia za skutočné dáta, teda napríklad názov nového súboru reportu, dátum a čas vygenerovania alebo odkaz na report v systéme. Rovnakým spôsobom som riešil aj

[11] `https://www.nuget.org/packages/FluentScheduler/`

posielanie mailových správ pri registrácii nového užívateľa, žiadosti o vyresetovanie hesla a pri informovaní administrátora o zlyhaní vygenerovania reportu. Na obrázku 36 je ukážka mailovej správy po vygenerovaní reportu a na obrázku 37 po registrovaní do systému administrátorom.



Obr. 36: Ukážka mailovej správy s odkazom na nový report (mobilná verzia)



Obr. 37: Ukážka mailovej správy po registrácii do systému (mobilná verzia)

4.7 HTTP Handler

Pre spracovanie individuálnych volaní koncového bodu sa používajú tzv. HTTP Handler-y. Na rozdiel od HTTP modulov, pre spracovanie volania je potrebný len jeden handler. Pre jeho vytvorenie je potrebné implementovať rozhranie `System.Web.IHttpHandler`. [25]

Na výpise 14 je ukážka implementácie handlera z tohto systému, ktorý má na starosti otváranie detailu konkrétneho reportu na základe jeho identifikátora. Využíva sa na presmerovanie užívateľa z mailovej správy o novom reporte na konkrétny report v systéme. Hlavný kód handlera je v metóde `void ProcessRequest(HttpContext context)`, ktorú predpisuje rozhranie. V metóde sa najskôr overí správnosť volanej URL. Potom, na základe toho, či je v prehliadači prihlásený užívateľ, je buď presmerovaný na detail reportu, alebo na prihlasovaciu obrazovku s nastaveným parametrom `ReturnUrl`, ktorý zabezpečí, že po prihlásení je automaticky presmerovaný na detail reportu. Na výpise 15 je ukážka konfiguračného súboru aplikácie so sekciou pre zaregistrovanie tohto handlera.

URL adresa, na ktorú reaguje handler má tvar https://report.glovis.cz/OpenReport?report={identifkátor}. Ak by sa ako identifikátor reportu v adrese využil primárny kľúč z databázy, ktorý

sa inkrementuje s každým novým záznamom, umožňovalo by to užívateľovi ľahko tento údaj v URL adrese upraviť na vyššiu/nižšiu hodnotu, čím by sa dostal na detail iného reportu. Preto som do entity `Report` doplnil parameter UID, ktorý je typu GUID. Jedná sa o sekvenciu 32 náhodných číslic a písmen, a s každým novým záznamom sa vygeneruje jeho nová náhodná hodnota. Pre ešte lepšie zabezpečenie som implementoval aj doplnkovú ochranu, ktorá pri načítavaní stránky s detailom reportu overí, či prihlásený užívateľ má k danému reportu prístup. Ak nie, je informovaný hláškou a tlačidlo pre stiahnutie je deaktivované.

```
public class ReportHandler : IHttpHandler {
    ...
    public void ProcessRequest(HttpContext context) {
        string report = context.Request.QueryString["report"];
        if (report == null || string.IsNullOrEmpty(report)) {
            context.Response.Redirect("~/NotFound.aspx");
        }
        HttpCookie authCookie = context.Request.Cookies[FormsAuthentication.
            FormsCookieName];
        if (!context.User.Identity.IsAuthenticated || authCookie == null) {
            context.Response.Redirect($"~/Default.aspx?ReturnUrl=~/User/
                ReportDetail.aspx?report={report}");
        } else {
            context.Response.Redirect($"~/User/ReportDetail.aspx?report={report}
                ");
        }
    }
}
```

Výpis 14: Ukážka HTTP handlera pre otváranie konkrétneho reportu

```
<location path="OpenReport">
    <system.webServer>
        <handlers>
            <add name="ReportHandler" path="OpenReport" verb="*" type="
                GlovisReport.Web.Handler.ReportHandler" />
        </handlers>
    </system.webServer>
</location>
```

Výpis 15: Ukážka súboru Web.config s konfiguráciou HTTP handlera

4.8 Logovanie udalostí

Pre logovanie udalostí som použil knižnicu *Apache log4net*[12]. Jedná sa o knižnicu, ktorá umožňuje presmerovať logovacie výpisy do rôznych cieľov. Vznikla ako port známeho frameworku *Apache log4j* na .NET. Zachoval si podobnosť s pôvodnou verziou, avšak s využitím výhod, ktoré poskytuje .NET. [26]

V prípade tejto implementácie bol požiadavok zadávateľa ukladať záznamy priamo do databázy, čo mi tento framework bez problémov umožnil. Konfigurácia logovania sa umiestňuje priamo do súboru `Web.config`. Tu sa pridala sekcia `log4net`, ktorá definuje pripojenie do databázy a tiež spôsob ukladania záznamov do nej. Na výpise 16 je ukážka časti konfiguračného súboru so sekciou pre logovaciu službu. V triede, kde chceme využiť logovanie, musíme najskôr získať inštanciu logovacej entity pomocou triedy `LogManager`. Následne, pre logovanie používame metódy ako napríklad `void Debug(object message), void Error(object message, Exception e)` a ďalšie. Ukážka použitia je na výpise 12 v metóde `void Execute()`.

```
<configSections>
    <section name="log4net" type="log4net.Config.
        Log4NetConfigurationSectionHandler, log4net" />
</configSections>
...
<log4net>
...
    <appender name="AdoNetAppender" type="log4net.Appender.AdoNetAppender">
        ...
        <connectionStringName value="MySqlConnection" />
        <commandText value="INSERT INTO log (DateTime,Thread,Level,Logger,
            Message,Exception) VALUES (?log_date, ?thread, ?log_level,?logger, ?
            message, ?exception)" />
        <parameter>
            <parameterName value="?log_date" />
            <dbType value="DateTime" />
            <layout type="log4net.Layout.RawTimeStampLayout" />
        </parameter>
        ...
    </appender>
</log4net>
```

Výpis 16: Ukážka súboru Web.config s konfiguráciou logovacej služby *Log4Net*

[12] `https://www.nuget.org/packages/log4net/`

5 Testovanie a nasadenie

V tejto kapitole sa zameriam na veľmi dôležitú fázu vývojového cyklu softvéru, ktorou je testovanie. Testuje sa za účelom overovania funkčnosti a kvality softvérového diela. Ďalej popíšem aj postup nasadenia aplikácie do produkcie. V prípade webovej aplikácie môžeme brať do úvahy tieto typy testovaní:

- Unit testovanie - overovanie funkčnosti komponent zdrojového kódu
- Testovanie užívateľského rozhrania (UI) - testovanie prezentačnej vrstvy, že správne zobrazuje dáta
- Výkonnostné testovanie - overovanie, ako degradujú kvalitatívne vlastnosti aplikácie s rastúcim množstvom záťaže

5.1 Unit testovanie

Pre potreby unit testovania som využil testovací open-source framework pre .NET *NUnit 3*, ktorý pôvodne vznikol z frameworku *JUnit* určeného pre jazyk Java. Tento framework poskytuje bohaté možnosti testovania a pokrýva širokú škálu .NET platforiem. Do projektu sa jednoducho integruje pomocou distribučnej platformy *NuGet*[13].

Pre každý projekt, pre ktorý som vytváral unit testy som si založil osobitný testovací projekt a pre každú testovanú triedu som vytvoril novú testovaciu triedu. Tieto triedy sa v *NUnit* označujú anotáciou `[TestFixture()]` a metódy `[Test()]`. Na výpise 17 je ukážka testovacej triedy pre triedu `StringUtils` z komponenty *Common*. Metóda, ktorá je testovaná, kontroluje silu hesla, teda či obsahuje minimálne 8 znakov, 1 číslicu, 1 veľký, 1 malý a 1 špeciálny znak. Použitý testovací framework podporuje parametrizované testy, čo umožňuje jednou testovacou metódou overovať rôzne vstupy. Zabezpečuje sa to anotáciou `[TestCase(params object[] arguments)]`, ktorej počet, poradie a typy parametrov zodpovedajú parametrom testovacej metódy. [27]

```
[TestFixture()]
public class StringUtilsTests {
    ...
    [Test()]
    [TestCase("", false)]
    [TestCase("aa", false)]
    [TestCase("aaaaaaaa", false)]
    [TestCase("AA", false)]
    [TestCase("AAAAAAAA", false)]
    [TestCase("11", false)]
    [TestCase("11111111", false)]
```

[13] `https://www.nuget.org/packages/NUnit/`

```
    [TestCase("aaAA11", false)]
    [TestCase("AAAAaaaa", false)]
    [TestCase("aaaa1111", false)]
    [TestCase("aA1", false)]
    [TestCase("TestPass123", false)]
    [TestCase("TtPs13_", false)]
    [TestCase("TestPass123_", true)]
    public void CheckStrongPasswordTest(string input, bool isValid) {
        Assert.AreEqual(isValid, StringUtils.CheckStrongPassword(input));
    }
}
```

Výpis 17: Ukážka testovacej triedy pre unit test

5.2 UI testovanie

Prezentačná vrstva by mala byť testovaná ako aj iné časti aplikácie. Je potrebné overiť správne zobrazovanie dát a tiež spracovanie vstupov od užívateľa, čo môže byť veľmi komplikované. Stav užívateľského rozhrania musí byť správne aktualizovaný na základe užívateľských vstupov.

Pre UI testy sa môže využiť rovnaký testovací framework ako pre unit testy, avšak potrebujeme k nemu doplnkový nástroj, ktorý dokáže pracovať s prvkami užívateľského rozhrania webovej stránky a ovládať ich. Ja som si zvolil nástroj *Selenium*, konkrétne *Selenium WebDriver*. Pomocou neho je možné vytvárať UI testy podobne ako unit. Rovnako ako v prípade *NUnit*, aj tento nástroj je možné integrovať pomocou *NuGet*. Ponúka množstvo spôsobov vyhľadania elementov na stránke a následným vykonávaním akcii nad nimi, a v spolupráci s *NUnit* tiež možnosť kontrolovať ich stav[28].

Pre čistejšiu implementáciu som si volania niekoľkých metód z testovacích frameworkov zaobalil do jednej pomocnej metódy. Napríklad metóda `void AssertTextOnElement(IWebDriver driver, By by, string text)` je v skutočnosti spojenie metódy zo *Selenium WebDriver* a *NUnit* frameworku a je zobrazená na výpise 18.

```
public class Utils {
    ...
    public static void AssertTextOnElement(IWebDriver driver, By by, string
        text) {
        var elementText = driver.FindElement(by).Text;
        Assert.AreEqual(text, elementText);
    }
}
```

Výpis 18: Ukážka pomocnej metódy pre UI testy

Testovacie prostredie

Keďže systém sa bude primárne využívať cez prehliadať *Google Chrome*, UI testy boli spúšťané na tomto prehliadači. Testovacie triedy a metódy majú rovnaké notácie ako v prípade unit testov. Hlavným rozhraním je `IWebDriver`, ktorého inštancia ovláda prehliadač, v ktorom test prebieha. Každý krok testu teda pracuje s touto inštanciou. Pre *Google Chrome* existuje jeho implementácia `ChromeDriver` a je nutné ju integrovať ako ďalší *NuGet* balíček. Pre čistejšiu implementáciu sa doporučuje, aby testovacia trieda implementovala rozhranie `IDisposable` a v metóde `void Dispose()` ukončila a uvoľnila inštanciu `IWebDriver`.

```
[TestFixture]
public class TestSecuredSystemAccess : IDisposable {
    private readonly IWebDriver _driver;
    public TestSecuredSystemAccess() { _driver = new ChromeDriver(); }
    ...
    [SetUp]
    public void SetupBeforeEachTest() {
        _driver.Manage().Cookies.DeleteAllCookies();
        _driver.Navigate().GoToUrl("https://report.glovis.cz/");
    }
    [Test(Description = "TC2")]
    [Category("UI_tests")]
    public void RegisteredUserLoginTest() {
        //Step 1
        Utils.PerformClick(_driver, By.Id("ContentPlaceHolder1_Button_login"));
        //Check step 1
        Utils.AssertTextOnElement(_driver, By.Id("
            ContentPlaceHolder1_Label_error"),Language.enter_login_credentials);
        //Step 2
        Utils.TypeText(_driver, By.Id("ContentPlaceHolder1_TextBox_mail"), "
            test");
        Utils.PerformClick(_driver, By.Id("ContentPlaceHolder1_Button_login"));
        //Check step 2
        Utils.AssertTextOnElement(_driver, By.Id("
            ContentPlaceHolder1_Label_error"),Language.enter_login_credentials);
        //Step 3
        Utils.TypeText(_driver, By.Id("ContentPlaceHolder1_TextBox_mail"), "
            testt@mail.com");
        Utils.TypeText(_driver, By.Id("ContentPlaceHolder1_TextBox_password"),
            "test");
```

```
        Utils.PerformClick(_driver, By.Id("ContentPlaceHolder1_Button_login"));
        //Check step 3
        Utils.AssertTextOnElement(_driver,By.Id("
            ContentPlaceHolder1_Label_error"),Language.user_with_mail_not_exist)
            ;
        ...
    }
}
```

Výpis 19: Ukážka testovacej triedy pre UI test

Testovacie scenáre

U každého testu užívateľského rozhrania musíme mať ujasnených niekoľko informácii, aby sme si udržiavali spätnú vysledovateľnosť v prípade, že test zlyhá. Tým si vieme ihneď dohľadať, ktoré funkčné požiadavky sa narušili a do ktorej oblasti biznis požiadaviek zasahujú. Všetky tieto potrebné informácie sa udržiavajú v štruktúre nazývanej testovací scenár. Existuje mnoho variant testovacích scenárov, avšak v základe by mal obsahovať nasledujúce informácie:

- Identifikátor,
- Názov,
- Funkčné/technické požiadavky, ktoré testuje,
- Stručný popis,
- Jednotlivé kroky testu a ku každému z nich aj očakávané výsledky.

V prípade tejto aplikácie nebolo možné automatickými testami pokryť úplne všetky funkčné a technické požiadavky. Tabuľka 6 zobrazuje vybrané testovacie prípady pre testy užívateľského rozhrania a s nimi previazané technické, funkčné a biznis požiadavky. Na obrázku 38 je ukážka testovacieho scenáru pre testovanie prihlasovania registrovaného užívateľa. Ako môžeme vyčítať zo scenára, v prípade, že tento test zlyhá, vieme na základe obsiahnutých informácii presne identifikovať, ktorý technický požiadavok už nebude splnený, a tiež ktorá oblasť požiadaviek je týmto zlyhaním zasiahnutá. Na výpise 19 je ukážka kódu UI testu, ktorý je označený anotáciou s popisom obsahujúcim číslo testovacieho prípadu, aby bolo možné vysledovať test z kódu až ku konkrétnemu požiadavku. V samotnom kóde testu je dobrou praxou si označiť jednotlivé kroky scenáru, pre lepšie dohľadanie, ktorý krok zlyhal.

Testovacie dáta

V prípade automatizovaného testovania je bežnou potrebou, aby sa pred, alebo po každom teste, prípadne pred alebo po celej sade testov vykonali niektoré kroky. Na to poskytuje *NUnit* anotáciu `[SetUpFixture]` pre triedu, ktorá obsahuje takýto kód. Metódy spúšťané pred, resp.

Tabuľka 6: Tabuľka vysledovateľnosti pre testovacie prípady užívateľského rozhrania

Biznis požiadavok	**Funkčný požiadavok**	**Technický požiadavok**	**Testovací prípad**	**Názov testovacieho prípadu**
BR1	FR1	FR1.2	TC1	Prihlásenie doménového užívateľa prihlasovacími údajmi
BR1	FR1	FR1.3	TC2	Prihlásenie bežného užívateľa
BR1	FR1	FR1.4	TC3	Rozlíšenie role užívateľa
BR1	FR3	FR3.1, FR3.2, FR3.3	TC4	Zmena dočasného hesla
BR1	FR4	FR4.1	TC5	Žiadosť o reset hesla
BR2	FR5	FR5.1, FR5.4	TC6	Registrácia užívateľa
BR2	FR6	FR6.1	TC7	Evidencia registrovaných užívateľov
BR3	FR7	FR7.1	TC8	Evidencia definícií reportov
BR3	FR13	FR13.1	TC9	Kontrola prístupu ku reportom
BR3	FR13	FR13.2, FR13.3	TC10	Filtrovanie reportov
BR4	FR9	FR9.4	TC11	Nastavenie prístupu ku reportom
BR5	FR16	FR16.1, FR16.2, FR16.3, FR16.4	TC12	Evidencia logovaných udalostí a ich filtrovanie
BR5	FR17	FR17.1	TC13	Detail logovanej udalosti
BR6	FR15	FR15.1	TC14	Manuálne vygenerovanie reportu

po každom teste sa anotujú `[OneTimeSetUp]`, resp. `[OneTimeTearDown]`. V mojom prípade som si týmto spôsobom vždy premazal testovaciu databázu a nanovo do nej vložil testovacie dáta, aby bolo zabezpečené, že každý test bude vždy pracovať s rovnakými dátami, a že vykonanie jedného testu neovplyvní nasledujúci. Takto anotované metódy sa automaticky vykonajú podľa potreby a nie je potrebné ich nikde explicitne volať.

Taktiež som potreboval zabezpečiť, aby test vždy začínal na prihlasovacej obrazovke. Aplikácia je však implementovaná tak, že ak je užívateľ uložený v cookies, tak je automaticky presmerovaný do systému a prihlasovacia obrazovka sa ani nezobrazí. Preto sa pred každým testom premažú cookies a presmeruje na prihlasovaciu obrazovku. Vo výpise 19 sa jedná o metódu `SetupBeforeEachTest()`, ktorá je v rámci testovacej triedy anotovaná `[SetUp]`.

5.3 Výkonnostné testovanie

Z povahy typu výslednej aplikácie môžeme usúdiť, že užívateľ bude so systémom pracovať v malej miere. V praxi to znamená, že užívateľ sa minimálne raz denne prihlási do systému, stiahne si nové reporty a systém opustí. Ako bolo spomínané v sekcii 2.1.1, reporty sa sprístupňujú len skupine zamestnancov s počtom približne 100 ľudí. Vezmime do úvahy, že v spoločnosti je niekoľko rôznych pozícii zamestnancov, pričom zamestnanci na rôznych pozíciach potrebujú pristupovať ku reportom, ktoré sa generujú v rôznych časoch. To znamená, že do systému nikdy nebude pristupovať naraz skupina maximálne 100 užívateľov, ale len zlomok z nich, odhadom 20-30. Takáto malá interakcia nepredstavuje výraznú záťaž na samotný systém. Z tohto dôvodu som sa po konzultácii so zastupujúcou osobou zo strany spoločnosti, pre ktorú je systém určený, rozhodol výkonnostné testovanie vynechať.

Testovací prípad TC2

Názov: TC2 - Prihlásenie bežného užívateľa

Biznis požiadavka: BR1 - Zabezpečený prístup do systému

Funkčná požiadavka: FR1 - Prihlásenie do systému

Technické požiadavky:

- FR1.3 - Systém bude schopný prihlásiť platného registrovaného užívateľa pomocou mailu a hesla

Popis: Test overuje správnu validáciu prihlasovacích údajov registrovaného užívateľa a zobrazovanie chybových hlášok. Kontrola zadaného formátu mailovej adresy. Kontrola prihlásenia len platného užívateľa.

Kroky:

Poradie	Akcia	Očakávané výsledky
1.	Kliknutie na tlačidlo "*Prihlásiť*"	Zobrazenie upozornenia, že je potrebné vyplniť prihlasovacie údaje
2.	Zadanie hodnoty "*test*" do poľa pre mail, kliknutie na tlačidlo "*Prihlásiť*"	Zobrazenie upozornenia, že je potrebné vyplniť prihlasovacie údaje
3.	Zadanie hodnôt Mail: "*testt@mail.com*" Heslo: "*test*" Kliknutie na tlačidlo "*Prihlásiť*"	Zobrazenie upozornenia, že užívateľ s daným mailom neexistuje
4.	Zadanie hodnôt Mail: "*test@mail.com*" Heslo: "*test*" Kliknutie na tlačidlo "*Prihlásiť*"	Zobrazenie upozornenia, o nesprávnom hesle
5.	Zadanie hodnôt Mail: "*test@mail.com*" Heslo: "*Test1234*" Kliknutie na tlačidlo "*Prihlásiť*"	Úspešné prihlásenie užívateľa. Na hlavnej lište sa nachádza meno užívateľa "*test@mail.com*"
6.	Kliknutie na tlačidlo "*Odhlásiť*"	Odhlásenie užívateľa, zobrazenie prihlasovacej obrazovky.
7.	Zadanie hodnôt Mail: "nonvalid*@mail.com*" Heslo: "*Test1234*" Kliknutie na tlačidlo "*Prihlásiť*"	Zobrazenie upozornenia, že užívateľ s daným mailom neexistuje

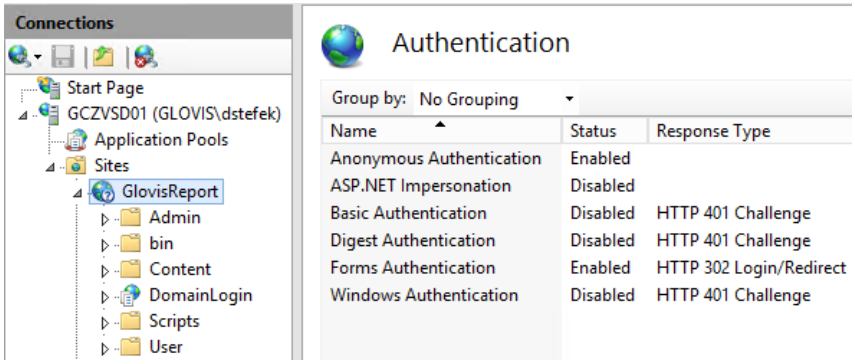
Obr. 38: Ukážka scenáru pre testovací prípad

5.4 Nasadenie

Aplikácia bola nasadená do produkčného prostredia pomocou nástroja IIS Manager na serveri s operačným systémom Windows Server 2012 Datacenter vo firemnej infraštruktúre. Pri nasadení sme chceli, aby bola aplikácia dostupná cez šifrovaný HTTPS protokol pod portom 443, čo vyžadovalo nastavenie firemného certifikátu.

Prihlasovanie

Ako bolo spomínané v sekcii 4.1, pre prihlasovanie bolo požadovaná *Windows* a *Forms Authentication*, ktoré však natívnym spôsobom nebolo možné implementačne skombinovať a preto som pristúpil k vlastnej implementácii. Pri nasadzovaní som narazil na podobný problém - na jednu aplikáciu nie je možné nastaviť tieto dva typy autentifikácie. Tento problém som vyriešil vytvorením novej rovnakej aplikácie vo vnútri existujúcej. Vonkajšej aplikácii som nastavil *Forms* a *Anonymous Authentication* (viď obrázok 39), vnútornej *Windows Authentication*. Vnútornej aplikácii som naviac upravil cestu na `https://report.glovis.cz/DomainLogin`. V praxi to teda funguje tak, že užívateľ sa pri požiadavke na adresu `https://report.glovis.cz` dostane na aplikáciu s formulárovým prihlasovaním bez potreby doplnkovej autorizácie, kde sa môže do systému prihlásiť ako registrovaný užívateľ. V prípade, že sa chce prihlásiť pomocou doménového účtu alebo automaticky cez windows účet v počítači, cez tlačidlo sa presmeruje na `https://report.glovis.cz/DomainLogin`. Na obrázku 39 je možné vidieť v adresárovej štruktúre vnorenú aplikáciu pre doménové prihlasovanie.



Obr. 39: Nastavenie spôsobov autentifikácie v aplikácii v nástroji IIS Manager

Záver

Praktickým výstupom tejto práce je funkčný informačný systém, implementovaný vo frameworku ASP.NET Web Forms. Je nasadený na serveri vo firemnej infraštruktúre a je dostupný všetkým zamestnancom na adrese `https://report.glovis.cz`, zatiaľ len v rámci internej siete. V súčasnom stave je systém schopný automaticky generovať reporty v stanovených intervaloch na základe SQL dotazu a podporuje .xlsx a .csv formáty výstupných súborov. Do systému majú prístup všetci zamestnanci spoločnosti *Hyundai Glovis Czech Republic s.r.o.* a tiež externé osoby podľa potreby. Užívatelia sú o vygenerovaní reportu informovaní mailovou správou.

Prehľad možností vývoja pomocou technológie .NET mi ukázal, že sa jedná o skutočne silný a bohatý nástroj, so širokou paletou cieľových platforiem. Framework ASP.NET podporuje rôzne spôsoby autentifikácie a autorizácie, takže v tejto oblasti pokryje všetky požiadavky. Aj keď v súčasnosti existuje mnoho iných a modernejších frameworkov a knižníc pre webový vývoj, framework ASP.NET Web Forms má stále čo ponúknuť. Ešte lepšie možnosti prináša v kombinácii so vzorom MVP.

Z prieskumu súčasného stavu som zistil všetky nedostatky, na ktoré bolo potrebné sa zamerať pri návrhu nového systému. Z môjho pohľadu bolo najväčším nedostatkom zbytočné plytvanie diskového priestoru reportami odosielanými v prílohách. Nový systém vyžaduje existenciu len jediného súboru pre daný report na serveri. Ako bolo spomenuté v sekcii 2.1.1, priemerná veľkosť jednej mailovej správy s reportom bola 40,7kB. Mailová správa s odkazom na nový report rozposielaná novým systémom má veľkosť približne 6,4kB, bez ohľadu na veľkosť reportu. Mesačná spotreba diskového priestoru je teda približne 1,53GB, čo je len 15,76% z pôvodnej veľkosti 9,71GB. Zložitá dohľadateľnosť reportov v množstve mailov v pôvodnom riešení je v novom systéme vyriešená jednoduchou evidenciou všetkých reportov na jednom mieste s možnosťou vyhľadávania a filtrovania. V neposlednom rade nový systém umožňuje novému užívateľovi ľahko sprístupniť potrebné reporty, vrátane tých, ktoré boli vygenerované ešte pred tým, než získal prístup do systému. Pokročilejšími funkciami knižnice *ClosedXML*, použitej pre generovanie .xlsx súborov je možné dosiahnuť formát reportu podobného ako na obrázku 10 na strane 31.

Z vykonaného prieskumu existujúcich riešení podobného zamerania som usúdil, že vzhľadom na jednoduchosť reportov, ktoré od nového systému zadávateľ očakával by kúpa takéhoto hotového riešenia nebola výhodná. Naviac, zadávateľ chcel reportovací systém ako samostatnú aplikáciu a väčšina dostupných riešení je ponúkaná ako komponenta do existujúcich systémov.

Pri návrhu systému som postupoval tak, že som si najskôr so zadávateľom vyjasnil, aké funkcie od nového systému požaduje. Tie som následne previedol na funkčné požiadavky a vytvoril prvotné návrhy užívateľského rozhrania. Nasledovalo vytvorenie logického modelu, kde som si navrhol, ako bude systém rozdelený do jednotlivých komponentov. V doménovom modeli som si utvoril predstavu o vzťahoch medzi hlavnými doménovými objektami. Najrozsiahlejšia časť celého návrhu bola venovaná analýze prípadov užitia. Tam som mal možnosť navrhnúť do detailu, ako budú prípady užitia, u ktorých je potrebná interakcia užívateľa, implementované,

vrátane návrhu kompletného užívateľského rozhrania. Takto detailne pripravený návrh mi výrazne uľahčil a urýchlil následnú implementáciu. U každého prípadu užitia som presne vedel, s ktorými doménovými objektami sa bude pracovať, aké akcie vyvolajú jednotlivé ovládacie prvky na stránke a tiež ktoré informácie budú zobrazené. V tejto analýze som pokryl všetky funkčné požiadavky pre všetky spracované prípady užitia.

Počas implementácie systému som za najväčší problém považoval skombinovanie rôznych typov prihlasovania, avšak výsledný systém poskytuje všetky požadované od zadávateľa. Vzor Model View Presenter som pre implementáciu prezentačnej vrstvy použil prvýkrát a môžem zhodnotiť, že sa implementuje veľmi jednoducho. Dátovú vrstvu je možné vďaka použitiu návrhových vzorov *Repository* a *Abstract Factory* vymeniť za inú podľa potreby.

Automatické testovanie užívateľského rozhrania umožňuje po prípadných ďalších úpravách systému rýchlo a jednoducho skontrolovať, či ostala zachovaná pôvodná funkčnosť systému. Vďaka dobre zavedenej vysledovateľnosti je možné jednoducho odhaliť, čo všetko je zmenou, prípadne zlyhaním zasiahnuté. Systém je nasadený pomocou nástroja IIS Manager, s ktorým majú administrátori v spoločnosti bohaté skúsenosti, takže v prípade potreby si môžu sami upraviť jeho konfiguráciu.

Systém je implementovaný tak, aby ho bolo možné v budúcnosti rozšíriť o ďalšie typy reportov. Keďže som pri implementácii využil vzor MVP, je možné v prípade potreby previesť systém na desktop aplikáciu, napríklad do frameworku WPF. So zadávateľom sme sa dohodli, že aj po odovzdaní tejto diplomovej práce budem ešte nejaký čas pracovať na vylepšovaní, stabilizovaní a prípadnom upravovaní a rozširovaní systému podľa nových požiadaviek.

Keďže som splnil všetky náležitosti zadania a výsledný systém poskytuje všetky funkcionality, ktoré požadoval zadávateľ, považujem zadanie tejto práce za splnené.

Literatúra

[1] *Informácia* [online]. [cit. 2020-04-30]. Dostupné z: http://pdf.truni.sk/e-ucebnice/usi/data/3369654e-2e03-4dd0-91b2-4cea893f87eb.html?ownapi=1

[2] *Informácie, údaje, dáta* [online]. 2019-02-21 [cit. 2020-04-30]. Dostupné z: https://www.euroekonom.sk/informacie-udaje-data/

[3] CHRISTENSSON, Per. *Web Application Definition* [online]. 2014-02-17 [cit. 2020-04-21]. Dostupné z: https://techterms.com/definition/web_application

[4] *Top 10 Web Development Frameworks of 2020* [online]. 2020-01-19 [cit. 2020-04-24]. Dostupné z: http://www.techgeekbuzz.com/web-development-frameworks/

[5] *Tour of .NET* [online]. 2017-05-22 [cit. 2020-04-21]. Dostupné z: https://docs.microsoft.com/sk-sk/dotnet/standard/tour

[6] *.NET architectural components* [online]. 2017-08-23 [cit. 2020-04-21]. Dostupné z: https://docs.microsoft.com/sk-sk/dotnet/standard/components

[7] *.NET Standard* [online]. 2020-02-13 [cit. 2020-04-21]. Dostupné z: https://docs.microsoft.com/sk-sk/dotnet/standard/net-standard

[8] *.NET Core overview* [online]. 2020-03-26 [cit. 2020-04-21]. Dostupné z: https://docs.microsoft.com/sk-sk/dotnet/core/about

[9] *Overview of .NET Framework* [online]. 2017-03-30 [cit. 2020-04-22]. Dostupné z: https://docs.microsoft.com/sk-sk/dotnet/framework/get-started/overview

[10] *ASP.NET overview* [online]. 2019-08-10 [cit. 2020-04-22]. Dostupné z: https://docs.microsoft.com/sk-sk/aspnet/overview

[11] MITCHELL, S. *Security Basics and ASP.NET Support (C#)* [online]. 2008-01-13 [cit. 2020-04-23]. Dostupné z: https://docs.microsoft.com/en-us/aspnet/web-forms/overview/older-versions-security/introduction/security-basics-and-asp-net-support-cs

[12] *ASP.NET security overview* [online]. 2018-04-19 [cit. 2020-04-23]. Dostupné z: https://support.microsoft.com/sk-sk/help/891028/asp-net-security-overview

[13] MONINGI, L. *Authentication and Authorization in ASP.NET* [online]. 2018-11-23 [cit. 2020-04-24]. Dostupné z: https://www.c-sharpcorner.com/article/authentication-and-authorization-in-Asp-Net/

[14] MITCHELL, S. *User-Based Authorization (C#)* [online]. 2008-01-18 [cit. 2020-04-24]. Dostupné z: https://docs.microsoft.com/en-us/aspnet/web-forms/overview/older-versions-security/membership/user-based-authorization-cs

[15] MITCHELL, S. *Role-Based Authorization (C#)* [online]. 2008-03-24 [cit. 2020-04-24]. Dostupné z: https://docs.microsoft.com/en-us/aspnet/web-forms/overview/older-versions-security/roles/role-based-authorization-cs#introduction

[16] *What is Web Forms* [online]. 2014-02-21 [cit. 2020-04-22]. Dostupné z: https://docs.microsoft.com/sk-sk/aspnet/web-forms/what-is-web-forms

[17] ESPOSITO, D. SALTARELLO, A. *Microsoft .NET: Architecting Applications for the Enterprise*. Washington: Microsoft Press, 2009, 304s. ISBN 073562609X.

[18] ŠÍPOŠ, Juraj. *Cron – správca úloh* [online]. 2007-10-15 [cit. 2020-01-18]. Dostupné z: https://www.linuxexpres.cz/praxe/cron-spravca-uloh

[19] BOCHMANN, Gregor. *Purpose and Structure of Requirements Specifications* [online]. 2010 [cit. 2019-11-19]. Dostupné z: https://www.site.uottawa.ca/~bochmann/SEG3101/Notes/SEG3101-ch3-2%20-%20Requirements%20documentation%20standards%20-%20IEEE830.pdf

[20] *Course Registration System - Software Architecture Document* [online]. 1999-03-21 [cit. 2019-11-19]. Dostupné z: https://www.ecs.csun.edu/~rlingard/COMP684/Example2SoftArch.htm

[21] *Principal and Identity Objects* [online]. 2017-03-30 [cit. 2020-04-20]. Dostupné z: https://docs.microsoft.com/en-us/dotnet/standard/security/principal-and-identity-objects

[22] *Entity Framework 6* [online]. 2016-10-23 [cit. 2020-04-18]. Dostupné z: https://docs.microsoft.com/en-us/ef/ef6/

[23] *ClosedXML/ClosedXML* [online]. 2020 [cit. 2020-04-19]. Dostupné z: https://github.com/ClosedXML/ClosedXML

[24] *fluentscheduler/FluentScheduler* [online]. 2019 [cit. 2020-04-16]. Dostupné z: https://github.com/fluentscheduler/FluentScheduler

[25] *INFO: ASP.NET HTTP Modules and HTTP Handlers Overview* [online]. 2019-06-10 [cit. 2020-04-20]. Dostupné z: https://support.microsoft.com/en-us/help/307985/info-asp-net-http-modules-and-http-handlers-overview

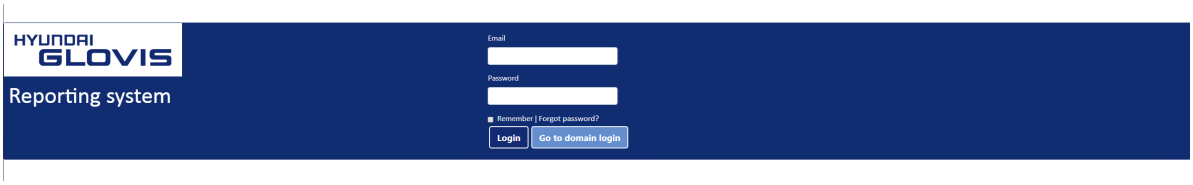
[26] *What is Apache log4net™* [online]. 2020-04-01 [cit. 2020-04-19]. Dostupné z: https://logging.apache.org/log4net/

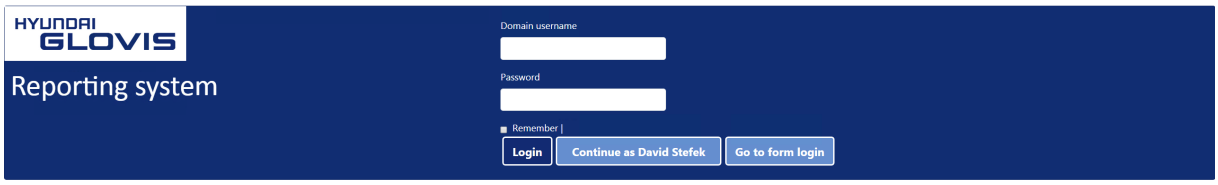
[27] *NUnit Documentation* [online]. 2016-08-03 [cit. 2020-04-11]. Dostupné z: https://github.com/nunit/docs/wiki/NUnit-Documentation

[28] *WebDriver* [online]. 2020 [cit. 2020-04-12]. Dostupné z: https://www.selenium.dev/documentation/en/webdriver/

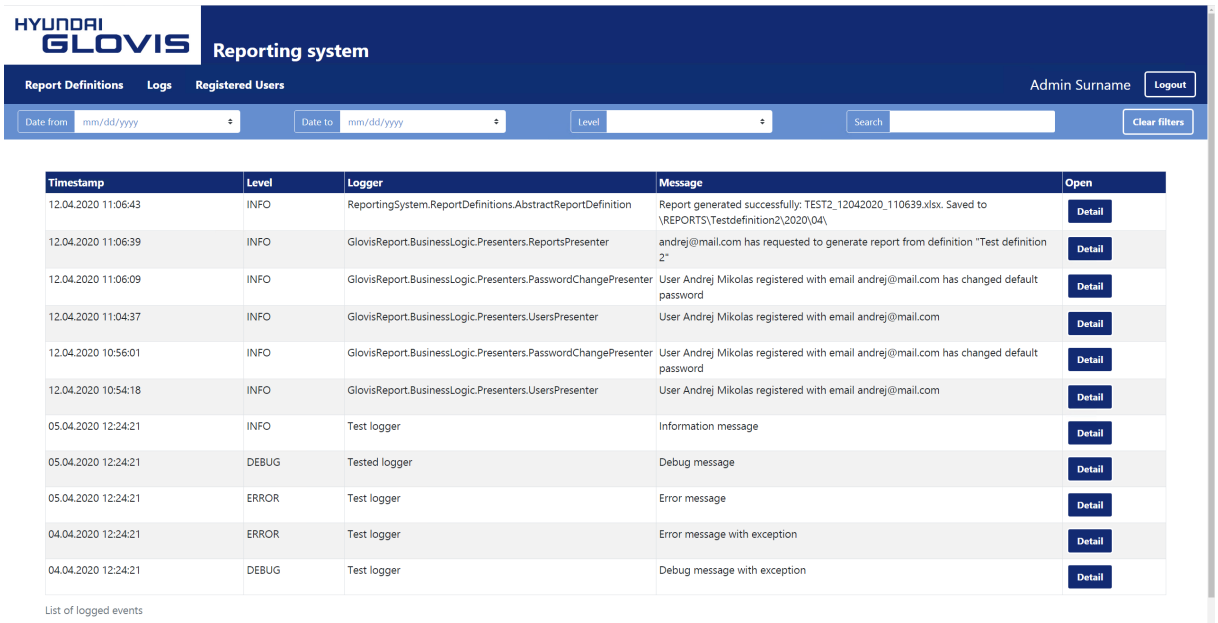
A Náhľad hotového užívateľského rozhrania

A.1 Spoločná časť

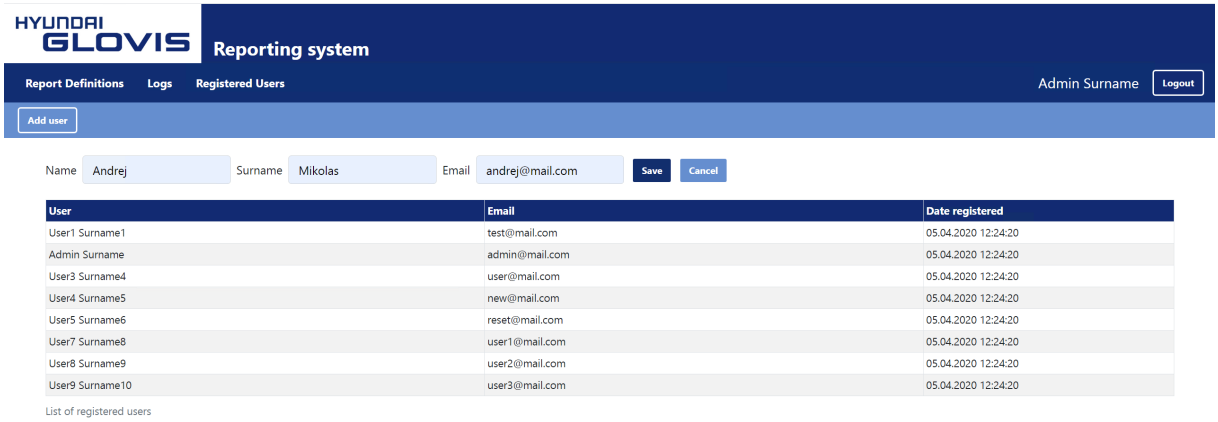




A.2 Administrátorská časť

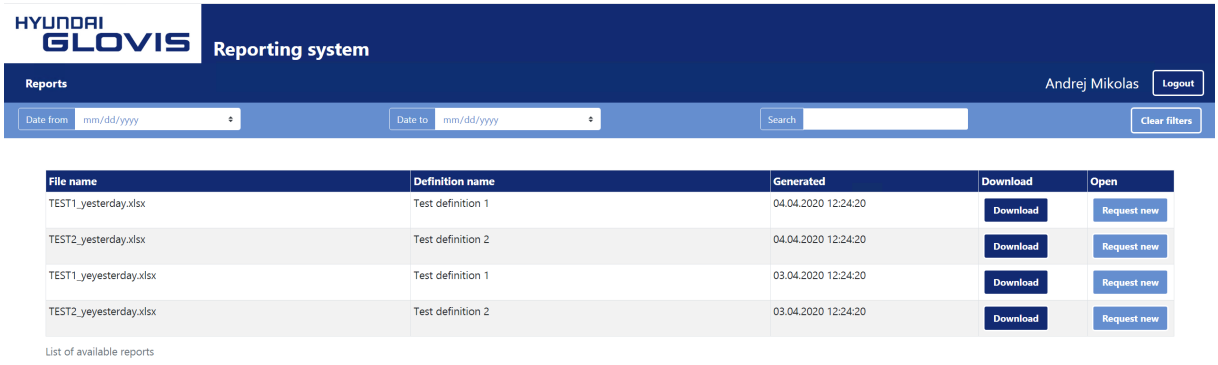


List of report definitions



List of registered users

A.3 Užívateľská časť



List of available reports